A forma correta de trabalhar com pesquisa
e lookups em aplicações cliente/servid

Clube Delphi +PHP

Turbine suas aplicações com AJAX Control Toolkit

Ano 8 · Edição 99 · R$9,90

DevMedia group

ISSN 1517990-7

NA WEB

Confira no portal ClubeDelphi PLUS 4 vídeo-aulas:

- Novidades do Delphi for PHP 2.0
- Busca incremental no ListBox
- Incorporando uma DLL ao executáve
- Destacando linhas no DBGrid

Mini-Curso
Datacar OO - Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 4

Boa Idéia
Capturando imagens e vídeos com o Delphi e criando um mecanismo de detecção de movimento

Firebird
Conheça as novidades da nova versão do servidor FB

Segurança
Conheça os novos controles de Login do ASP.NET 2.0

PHP
Paginação em Web Sites

Navegação
MasterPages, SiteMap, Menu e TreeView no ASP.NET 2.0

10 Motivos para hospedar seu site na RedeHost!

30 dias GRÁTIS para testar.

Mais de 15 mil clientes e 6 anos no mercado.

Até 50Gb de espaço para e-mails ou site.

Planos a partir de R$ 13,90.

Webmail em Ajax com calendário, tarefas, anotações e RSS.

ASP, PHP, .net e MySQL no mesmo plano.

Domínios adicionais ilimitados (apontam para o site principal).

Data center no Brasil com Links Redundantes.

Infra-estrutura com servidores Dell e redes Cisco.

Estatísticas de acesso com 75 relatórios.

RedeHost

www.redehost.com.br

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi

Web Novidades

06 – AJAX Control ToolKit
Turbine suas aplicações Web com ASP.NET 2.0
[*Maikel Scheid e Thiago Pasieka*]

Web Novidades

14 – Login
Veja como é simples utilizar os novos controles de Login do ASP.NET 2.0
[*Guinther Pauli e Adriano Santos*]

Web Novidades

24 – Navegação
Trabalhando com Menus, SiteMap, MasterPages e TreeView no ASP.NET 2.0
[*Guinther Pauli e Adriano Santos*]

Boas Práticas Mini-Curso e Expert

28 – Sistema DataCar em OO
Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 4
[*Paulo Quicoli*]

Boa Idéia

38 – Windows Vídeo API
Manipulando vídeo e imagem no Windows
[*Thiago Batista Limeira*]

Banco de Dados

46 – Novidades do Firebird 2.1
Veja o que há de mais interessante na nova versão do banco de dados Firebird
[*Adriano Santos*]

Easy Delphi

Easy Delphi

54 – Pesquisas e Lookups incrementados
Saiba a maneira correta de realizar pesquisas e smart lookups no seu BD
[*Guinther Pauli*]

PHP

PHP

60 – Paginação em Web Sites PHP
Veja como paginar os resultados de suas buscas no banco de dados com PHP
[*Adriano Santos*]

Você percebeu os nomes ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Mini-Curso] Artigo que faz parte de um mini-curso.

[PHP] Artigo com foco no PHP puro.

[Boas Práticas] A Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas, mas também as melhores práticas – Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre.

[Boa Idéia] A Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma 'boa idéia', dentro do contexto de desenvolvimento de software.

[Banco de Dados] Artigo sobre o universo de persistência de dados.

Ano 8 - 99ª Edição - 2008 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Vinicius O. Andrade
viniciusoandrade@gmail.com

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

A febre do momento é o desenvolvimento com ASP.NET 2.0 e AJAX no RAD Studio 2007. Se você ainda não criou nenhuma aplicação com essa tecnologia, aproveito para dizer que é tão simples que dá até para usar este editorial para explicar: no RAD 2007 inicie um novo AJAX-Enabled ASP.NET Web Application, largue no formulário um ScriptManager e logo abaixo um UpdatePanel. Programe sua interface dentro do UpdatePanel e pronto! Tudo é feito sem refresh total de página, com updates parciais e chamadas assíncronas XML-HTTP.

Podemos dizer que aplicações Web são semelhantes a aplicações Desktop. Será o fim das aplicações Desktop? Não ouso em dizer que em muito breve você pode ter um computador sem sistema operacional e só um browser.

E como se já não bastasse, para incrementar ainda mais o poder do ASP.NET AJAX, temos uma poderosa e gratuita suíte de componentes para AJAX, conhecida como AJAX Control ToolKit, destinada a ser utilizada em aplicações com ASP.NET 2.0 e 3.5. Se você acha que agora aplicações Web podem ser semelhantes a aplicações Desktop, agora você não terá mais dúvidas! É incrível a variedade de controles que essa suíte, quase que obrigatória no desenvolvimento com AJAX, proporciona. São controles para acabar com as limitações do HTTP e da Web: janelas popup modais, calendários e controles popup e dropdown, efeitos, animações, máscaras, validações incrementadas, inputs para senhas com indicador de complexidade, verificador de bots, auto complete, filtragem de texto e meu preferido, um PageControl para Web! Isso mesmo, aquilo que o HTML nunca teve (a não ser com simulações com imagens) agora disponível para AJAX, tudo assíncrono e com update parcial. Nesta edição, o Maikel e o Tiago dão uma introdução a esse poderoso framework para AJAX, o AJAX Control ToolKit.

E já que a febre é ASP.NET 2.0, temos muito sobre a tecnologia nesta edição: aprenda a utilizar os novos controles de Login para garantir a segurança do seu Web Site. Outro recurso também muito simples de ser utilizado, parte do ASP.NET 2.0, diz respeito à navegação: use MasterPages, SiteMaps, Menus e TreeViews para criar a estrutura de navegação do seu Web Site. O Paulo Quicoli finaliza o DataCar Orientado a Objetos, cumprindo a sua missão, de trazer o mundo OO ao universo Delphi. O Thiago apresenta um excelente artigo em uma boa idéia: como capturar imagens e vídeos em Delphi, mostrando ainda como criar um mecanismo de detecção de movimento. Interessantíssimo.

Sobre Banco de Dados, temos as novidades do Firebird 2.1, por Adriano Santos. E o Maikel apresenta como trabalhar com os metadados do banco de dados. Na sessão Easy aprenda com o editor como criar pesquisas e lookups de forma correta em aplicações cliente/servidor. E para finalizar, confira nossa sessão PHP!

Sucesso com o Delphi!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD
Delphi Certified: 6, 7, 2005, 2006, Web, Kylix

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Assinatura
ClubeDelphi PLUS
Mais conteúdo Delphi por menos

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+600 vídeo aulas e 7 cursos online

Brinde na web desta edição

4 Vídeos

Delphi for PHP – Novidades do PHP 2.0
Veja todas as novidades da nova versão do Delphi for PHP. O colunista Rodrigo Mourão mostra tudo que há de mais interessante na nova versão do Delphi for PHP.
http://www.devmedia.com.br/articles/viewcomp.asp?comp=9580

Criando uma busca incremental em um ListBox
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8330

Incorporando uma DLL ao executável
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8393

Destacando as linhas do DBGrid ao mover o mouse
http://www.devmedia.com.br/articles/viewcomp.asp?comp=8335

Para acessar os vídeos utilize os seguintes dados: Login: **DVM.PLS** Senha: **IWW*IL**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link: *www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:

Login: **DVM.PLS** Senha: **IWW*IL**

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

AJAX Control ToolKit

Turbine suas aplicações Web com ASP.NET 2.0

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Tiago José Pasieka
(tiagopasieka@gmail.com)
é técnico em informática, Analista e Programador de Sistemas com Delphi .NET, banco de dados Firebird e SQL Server.

Resumo DevMan

O AJAX Control ToolKit é uma poderosa suíte de componentes para serem utilizadas em aplicações Web que utilizem ASP.NET. Inclui componentes para otimizar validações, calendários, controles da abas, janelas popups, animações e muito mais.

Nesse artigo veremos

• Utilização de Ajax em aplicações ASP.NET;
• Utilização da suíte de componentes Ajax ToolKit.

Qual a finalidade

Entender como e onde usar Ajax em aplicações Web.

Quais situações utilizam esses recursos?

Qualquer Web Site que necessite de velocidade de acesso e usabilidade.

Tornando cada vez mais indispensável em aplicações web, o uso de tecnologias AJAX torna páginas de sites e sistemas Web mais dinâmicas e mais rápidas. Um exemplo prático a ser citado é a atualização de uma página pela segunda vez, ou melhor, depois de carregada completamente a página, o usuário pede um serviço ao servidor e ao retornar a requisição, a atualização da página acontece somente no que há de diferente em relação ao que já tinha, não precisando atualizar toda ela, deixando assim bem mais rápida a navegação.

Com esse grande avanço do *AJAX* é que surgiu mais uma nova ferramenta para desenvolvimento de aplicações, é o *AJAX Control Toolkit*. Um conjunto de ferramentas para aplicações *ASP.NET* com suporte *AJAX* que foi desenvolvido em conjunto entre a comunidade e a Microsoft para versões do *Framework* acima do 2.0 com *ASP.NET AJAX Extensions 1*.

Baseando-se em *AJAX*, uma das principais novidades do *Framework 2.0*, nesse artigo mostraremos a instalação e alguns controles do *AJAX Control Toolkit*, também como realizar as customizações desses componentes, tais como *CalendarExtender*, calendário de busca de data para adicionar a um *Tex-*

tBox, o *MaskEditExtender*, responsável por aplicar máscaras no *TextBox* e o *ValidatorCalloutExtender* responsável por exibir as mensagens dos componentes da *Web Validation*.

Instalando o AJAX Control Toolkit

Para instalação do pacote de componentes *AJAX ToolKit*, faça o download no endereço *www.codeplex.com/AjaxControlToolkit/Release/ProjectReleases.aspx?ReleaseId=11121*, salve em alguma pasta na sua máquina e descompacte o arquivo de instalação. Localize na pasta *[...]\SampleWebSite\bin* a dll *AjaxControlToolKit.dll*, a qual é necessária para a instalação do pacote, copiando-a para o seguinte diretório *C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727*, local onde encontram-se todas as dlls da versão do Framework, pois se você excluir todo aquele pacote junto com a dll depois de o instalar, vai ocorrer o erro de que não está encontrando o caminho, que foi especificado no momento da instalação. Para que isso não aconteça, instale a partir dessa pasta.

Na instalação do componente, usaremos o *RAD 2007*, onde criaremos uma nova aplicação com suporte AJAX através do menu *File|New>Other>Delphi for .Net Projects>AJAX-Enable ASP.NET Web Application* definindo o nome do projeto como *AjaxToolKit* e no item *Location* selecione o diretório destino onde os arquivos do projeto serão criados. No item *Server* defina o servidor de aplicação para a hospedagem do sistema **(Figura 1)**. Sendo uma aplicação com suporte *AJAX*, as páginas *Default* já serão exibidas contendo dois componentes: o *Script Manager* e o *UpdatePanel*, que são os dois componentes responsáveis pelo funcionamento e atualização do *AJAX*. Ao criar uma nova página, os componentes deverão ser adicionados manualmente, arrastando-os da paleta de componentes.

Para a instalação do pacote de componentes, no menu *Component>Installed .Net Component* abra o gerenciador de componentes .NET (**Figura 2**), onde no campo *Category* digite uma identificação para o pacote *AjaxControlToolkit* e clique no botão *Select an Assembly* localizando a dll de instalação que armazenamos a pouco no diretório de pacotes do *Framework 2.0*. Ao selecionar a DLL, toda a lista de componentes a ser instalada será listada, bastando confirmar o gerenciador para que os mesmos estejam disponíveis na *Tool Palette* (**Figura 3**).

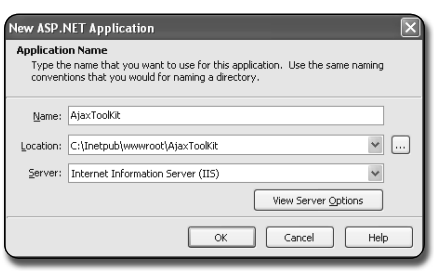


Figura 1. Registrando a Aplicação

Nota do DevMan

AJAX (acrônimo em língua inglesa de Asynchronous Javascript And XML) é o uso sistemático de tecnologias providas por navegadores, como Javascript e XML, para tornar páginas mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações web mais dinâmicas e criativas. AJAX não é uma tecnologia, são realmente várias tecnologias conhecidas trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. AJAX incorpora em seu modelo.:

* Apresentação baseada em padrões, usando XHTML e CSS;
* Exposição e interação dinâmica usando o DOM;
* Intercâmbio e manipulação de dados usando XML e XSLT;
* Recuperação assíncrona de dados usando o objeto XMLHttpRequest;
* e JavaScript unindo todas elas em conjunto.

O modelo clássico de aplicação web trabalha assim: A maioria das ações do usuário na interface dispara uma solicitação HTTP para o servidor web. O servidor processa algo — recuperando dados, realizando cálculos, conversando com vários sistemas legados — e então retorna uma página HTML para o cliente. É um modelo adaptado do uso original da Web como um agente de hipertexto, porém o que faz a Web boa para hipertexto não necessariamente faz ela boa para aplicações de software.
Esta aproximação possui muito dos sentidos técnicos, mas não faz tudo que um usuário experiente poderia fazer. Enquanto o servidor está fazendo seu trabalho, o que o usuário estará fazendo? O que é certo, esperando. E a cada etapa em uma tarefa, o usuário aguarda mais uma vez.

Obviamente, se nós estivéssemos projetando a Web a partir do zero para aplicações, não faríamos com que os usuários esperassem em vão. Uma vez que a interface está carregada, por que a interação do usuário deveria parar a cada vez que a aplicação precisasse de algo do servidor? Na realidade, por que o usuário deveria ver a aplicação ir ao servidor toda vez?

A maior vantagem das aplicações AJAX é que elas rodam no próprio navegador web. Então, para estar hábil a executar aplicações AJAX, bastar possuir algum dos navegadores modernos, ou seja, lançados após 2001. São eles: Mozilla Firefox, Internet Explorer 5+, Opera, Konqueror e Safari.

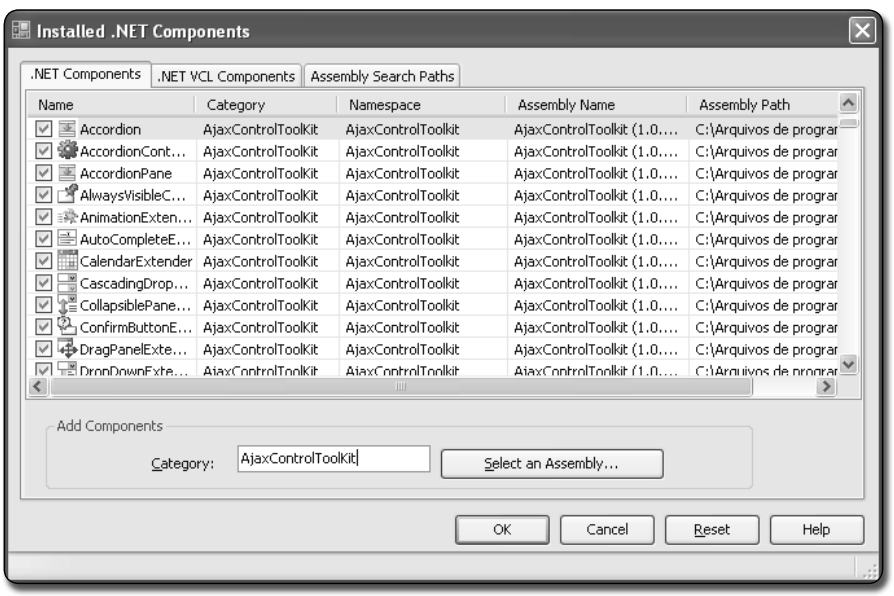


Figura 2. Instalando pacote de componentes AJAX ToolKit

Conhecendo e utilizando os componentes AJAX ToolKit

Antes de começar a adicionar componentes vamos aprimorar a aplicação, altere a página inicial, *Default.aspx* para *ajax.aspx*. Para isso clique com o botão direito do mouse no nome da página no *Project Manager* e selecione *Rename*. Antes de trabalharmos na página criada, iremos definir uma estrutura padrão para o cabeçalho do nosso sistema através de *User Control* na qual mais adiante faremos também o controle de usuários *logados*. Para adicionar o recurso é necessário criar uma página *.ascx* através do menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET User Control*. Dessa forma o Delphi fará a criação de uma nova página de edição do conteúdo. Altere no *Project Manager* o nome do *User Control* para *uctop.ascx*.

Nessa *UserControl* (*uctop.ascx*), abra o código pressionando a tecla *F12*, e no evento da *procedure Page_Load* adicione o código a seguir:

```
Page.Title := '
  Trabalhando com Ajax Control Toolkit';
```

Esse termo definirá um título padrão para todas as páginas que tiverem essa *User Control*, não precisando assim definir em cada página. Se quiseres, pode criar um rodapé, da mesma forma como criamos o cabeçalho, mas não precisando colocar o código *Page.Title* na *procedure Page_Load*.

Para aplicar as *User Control's* na sua página *ajax.aspx*, na janela *Project Manager*, procure o arquivo de *uctop.ascx*, selecione-o e arraste para a página acima do componente *UpdatePanel*, pois só ficarão dentro dela os controles da página. Se você criou a *User Control* representando o rodapé, siga os mesmos passos para arrastá-lo à página como rodapé da mesma.

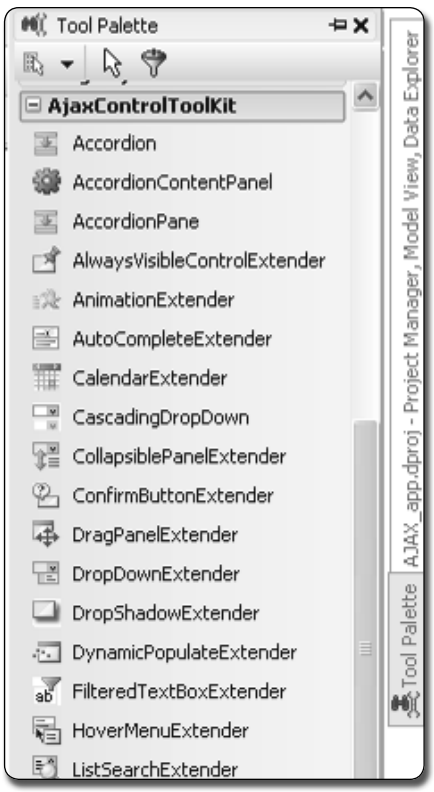


Figura 3. Paleta com componentes do pacote AJAX ToolKit

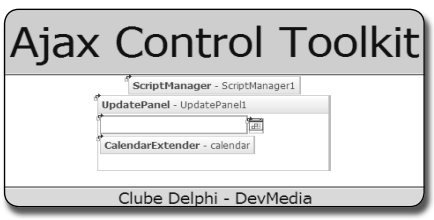


Figura 4. Adicionando o CalendarExtender à página

CalendarExtender

Antes de adicionar o componente *CalendarExtender* na página, através do *Windows Explorer*, procure a pasta onde são salvos os arquivos do projeto da aplicação (campo *location* da **Figura 1**) e crie uma nova pasta com o nome de imagens, pasta onde serão colocadas todas as imagens para a aplicação. Aproveitando, adicione nessa pasta um ícone que represente um calendário, que ocuparemos depois.

Agora no *Delphi*, dentro da *UpdatePanel*, da *Tool Palette* adicione da paleta *WebControl* os componentes *TextBox(txbData)* e ao lado dele um *ImageButton*(*imgbData*). No *ImageButton* adicione o ícone de calendário, através da janela *Object Inspector*, na propriedade *ImageUrl*, procure o ícone na pasta imagens que criamos e adicionamos anteriormente. Depois adicione da paleta *AjaxControlToolKit* que adicionamos no início do artigo, o componente *CalendarExtender*, em qualquer parte do *design*, mas dentro do *UpdatePanel* (**Figura 4**).

> **Nota:** A maioria dos componentes do *Toolkit* não precisa, necessariamente, estar ao lado do seu controlador, pois quando adicionados, no Delphi eles ficam ocupando espaço, mas quando executados no browser, eles ficam invisíveis, não percebendo que existe o componente.

Adicionado o componente de calendário, vamos configurar algumas propriedades na janela *Object Inspector* do *CalendarExtender*. Selecionando o componente, acesse suas propriedades e altere-as conforme configurações a seguir:

TargetControlID: aqui você digitará o ID do *TextBox* que irá receber a data (Ex: *txbData*). Este *TextBox* ao receber o foco do cursor exibirá automaticamente uma máscara de data que definiremos como componente de calendário;

*ID***:** calendar, nome que será atribuído ao componente;

Format: o formato é uma máscara de formatação para a data ao ser digitada ou selecionada no *TextBox* (Ex: dd/MM/yyyy);

FirstDayOfWeek: Selecionará qual será o primeiro dia da semana que será exibido no calendário(Ex: *Sunday*, semana irá começar sempre no dia de domingo).

PopupButtonPosition: Digitará o ID dado ao *ImageButton* (Ex: imgbData). É o componente que deverá ser acionado para que o calendário seja exibido. Ao clique do botão *imgbData* o calendário ficará visível.

*PopupPosition***:** Selecione uma posição para a exibição do calendário (Ex: *Rigth*). É a posição em relação ao componente de foco que o calendário irá assumir.

Realizadas as principais configurações necessárias para o funcionamento do componente, antes de ver o seu resultado (**Figura 5**), verifique se no código *.aspx* da sua página, a *tag HEAD* está como *runat:"server"* conforme código a seguir. Realizada a verificação do trecho de código, execute sua página e clique sobre o ícone do calendário adicionando, exibindo-o.

```
<HEAD runat="server">
  <TITLE></TITLE>
</HEAD>
```

Uma outra opção de visualização disponível seria se você não quisesse exibir o calendário ao clique do mouse sobre o botão, mas sim quando o foco for jogado ao *TextBox* por clique ou *tab* na página, bastando deixar em branco a propriedade *PopupButtonPosition*. Perceba também ao executar a aplicação que o calendário está escrito em inglês e você gostaria de visualizá-lo em português, como solucionar isso? Simples, no *Delphi*, no *design* da sua página, selecione o componente *ScriptManager* e altere para *True* as propriedades *EnableScriptGlobalization* e *EnableScriptLocalization*. Depois abra o seu arquivo *Web.Config*, na janela *Project Manager*, procure a *tag <globalization>* e adicione dentro ao seu corpo os seguintes termos, fazendo com que o idioma de apresentação do calendário seja em *Pt-BR* (**Figura 6)**:

```
culture="pt-BR" uiCulture="pt-BR"
```

MaskEditExtender

Lembra a máscara de data que aplicamos em nossos sistemas *Win32* geralmente utilizando o componente *MaskEdit* ou algum *DataSet* ligado a uma base de dados onde aparecem as barras pré-digitadas no componente?

Este mesmo recurso podemos utilizar agora na *web*, em nossas aplicações *for ASP.NET*. Aplicaremos através do componente *CalendaExtender* vinculado ao *TextBox*, uma máscara para quando o usuário não quiser selecionar uma data, mas sim digitá-la. Para isso, arraste da paleta *AjaxControlToolKit* o componente *MaskEditExtender* abaixo do componente *CalendaExtender* e altere as seguintes propriedades:

TargetControlID: Selecione o *TextBox* a ser controlado e deseja aplicar a máscara de data;

As duas próximas propriedades a serem adicionadas deverão ser incorporadas por meio de código diretamente no código *aspx* da página (**Figura 7**), onde deverá identificar o início das *tags* de criação do componente e adicionar as seguintes configurações (**Listagem 1**):

MASK: Formato do controle da máscara (Ex: 99/99/9999);

MASKTYPE= Qual é o tipo da máscara para o controle (Ex:Date).

O código adicionado fará com que, quando está executando a aplicação e posicionar com ponteiro dentro do *TextBox*, o formato correto de digitação da data será exibido, permitindo digitar somente números (**Figura 8**).

Listagem 1. Propriedades do MaskEdit

```
<cc1:MaskedEditExtender
  ID="MaskedEditExtender1"
  RUNAT="server"
  TARGETCONTROLID="txbData"
  MASK="99/99/9999"
  MASKTYPE="Date"
></cc1:MaskedEditExtender>
```

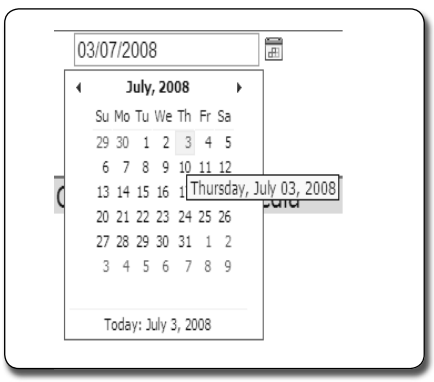


Figura 5. Calendário Padrão

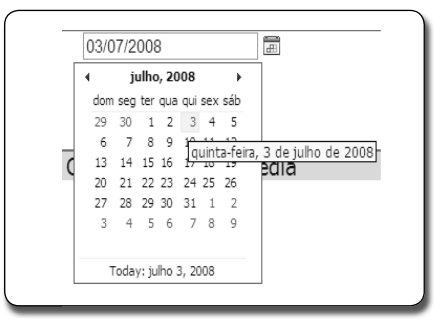


Figura 6. Calendário em Português

```
<asp:UpdatePanel id="UpdatePanel1" runat="server">
    <ContentTemplate>
      <asp:TextBox ID="txbData" RUNAT="server" WIDTH="167px"></asp
      <asp:ImageButton ID="imgbData" RUNAT="server" IMAGEURL="~/IM
      <asp:RegularExpressionValidator ID="revData" RUNAT="server"
                                      VALIDATIONEXPRESSION="^((((
                                      ERRORMESSAGE="Data Inválida"
      <cc1:CalendarExtender ID="calendar"
      RUNAT="server" CSSCLASS="calendar"
      FORMAT="dd/MM/yyyy" POPUPPOSITION="BottomRight"
      POPUPBUTTONID="imgbData" TARGETCONTROLID="txbData">
      </cc1:CalendarExtender>
      <cc1:MaskedEditExtender ID="MaskedEditExtender1"
      RUNAT="server" TARGETCONTROLID="txbData" MASK="99/99/9999"
      MASKTYPE="Date"></cc1:MaskedEditExtender>
```

Figura 7. Localização da Tag do component MaskEdit

MASKTYPE	MASK	Descrição
Date	99/99/9999	Para data
None	999,999,999-02	Para CPF
None	99,999-999	Para CEP
Time	99:99:99	Para Hora
DateTime	99/99/9999-99:99	Para Data e Hora
Number	9999	Para Números

Tabela 1. Tipos de Máscaras

A **Tabela 1** mostra mais alguns tipos de máscaras que poderão ser configuradas no componente *MaskEditExtender*.

ValidatorCalloutExtender

Outro recurso bastante útil para nossas aplicações é a validação das datas digitadas, verificar se o valor foi digitado corretamente, onde poderemos fazer o controle por meio do *RegularExpressionValidator* que é nativo do *ASP.NET*, responsável por validar a data informada e o *ValidatorCalloutExtender* do *Ajax Control Toolkit*, responsável por exibir a mensagem do *RegularExpressionValidator* em caso de data inválida. Adicione os dois componentes abaixo do *MaskEditExtender* e configure o componente *RegularExpressionValidator* (*revData*) conforme propriedades listadas a seguir:

ErroMessage: Digite uma mensagem de erro(Ex: Data Inválida);

Display: Selecione a opção *None*, ou seja, opção que não permitirá exibir a mensagem de erro no próprio componente;

ControlToValidate: Selecione o *txbData*;

ValidationExpression: coloque a seguinte expressão para que seja feita a validação de data no formato *"dd/MM/yyyy"*;

^(((((0[1-9])|(1\d)|(2[0-8]))\/((0[1-9])|(1[0-2])))|((31\/((0[13578])|(1[02])))|((29|30)\/((0[1,3-9])|(1[0-2]))))\/((20[0-9][0-9])|(19[0-9][0-9])))|((29\/02\/(19|20)(([02468][048])|([13579][26]))))$

Para o componente *ValidatorCalloutExtender*, a única propriedade a ser configurada é a *TargetControlID*, onde seleciona-se qual é o *RegularExpressionValidator* que o mesmo deverá controlar. Sempre lembrando que os componentes devem ficar dentro do *UpdatePanel*. Compile e veja o resultado (**Figura 9**).

Figura 8. Resultado do MaskEditExtender

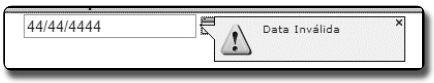


Figura 9. Resultado do ValidatorCalloutExtender ao digitar uma data inválida

Listagem 2. Estilo css do CalendarExtender

```
.calendario.ajax__calendar_container
/*formatação da parte externa onde encontra-se o corpo do calendário */
{background-color:#E6E6FA; border:solid 1px #004000;}
.calendario.ajax__calendar_header
/*formatação do cabeçalho*/
{background-color:#ffffff; margin-bottom:3px; border:solid 1px #004000; width:170px;}
.calendario.ajax__calendar_title
/*formatação do texto que encontra-se no cabeçalho*/
{color:black;}
.calendario.ajax__calendar_body
/*formatação interna de onde se encontra o calendário(onde estão os dias do respectivo
mês)*/
{background-color : white ;border : solid 1px #004000 ;}
.calendario.ajax__calendar_dayname
/*formatação da descrição dos dias da semana*/
{text-align : center ; font-weight : bold ; margin-bottom : 4px ;margin-top : 2px ;color
: red;}
.calendario.ajax__calendar_day
/*formatação dos dias do mês*/
{text-align : center ;color: black;}
.calendario.ajax__calendar_hover .ajax__calendar_day
/*formatação para quando passar o mause sobre um determinado dia*/
{color: red;}
.calendario.ajax__calendar_hover .ajax__calendar_month
/*formatação para quando passar mause sobre um determinado mês*/
{color: red;}
.calendario.ajax__calendar_hover .ajax__calendar_year
/*formatação para quando passar o mause sobre um determinado ano*/
{color: red;}
.calendario.ajax__calendar_active
/*formatação do dia, quando já possui uma data no textbox de controle*/
{background-color : #D4D0C8 ;font-weight : bold ; }
.calendario.ajax__calendar_today
/*destaque para a data atual(Today)*/
{ font-weight : bold ;color : red;}
.calendario.ajax__calendar_other .ajax__calendar_day
/*formatação dos dias que não pertencem ao mês selecionado*/
 {color:#6A5ACD;}
.calendario.ajax__calendar_hover .ajax__calendar_today
/*formatação para quando passar o mouse sobre a data atual(Today)*/
{color: black}
.calendario.ajax__calendar_hover .ajax__calendar_title
/*formatação para quando passar o mouse sobre o texto do título*/
{color : red ; }
```

Customizações com CSS

Depois de aplicados os componentes ao nosso formulário, podemos ainda observar que seus estilos estão em padrões pouco variados dos que costumamos ver em aplicações atuais, tais como cores e formas de apresentação. Da mesma forma como é possível aplicar estilos em componentes *web*, poderemos utilizar do recurso para os componentes da paleta *Ajax Control ToolKit*. Criaremos estilos variados que serão aplicados aos componentes *CalendarExtender* e o *ValidatorCalloutExtender* alterando sua forma de apresentação *default*.

Para início da construção dos estilos, crie uma nova pasta chamada "estilos", dentro da pasta do projeto da aplicação no mesmo nível de diretório onde criamos a pasta imagens anteriormente. Em seguida faremos uso do bloco de notas para criação de um novo arquivo, ao qual deverá ser adicionado o código da **Listagem 2** que encontra-se comentado em alguns trechos de acordo com a funcionalidade das linhas de código relacionadas.

Todas estas linhas de código serão responsáveis pela aparência dos nossos componentes *web*. Salve o arquivo dentro dessa pasta estilos recém criada com o nome de *estilos.css*.

Observe que os estilos criados anteriormente independem da linguagem de *ASP.NET*. Estamos referenciando a cada uma das funcionalidades do calendário a confecção de um estilo, aplicando a ele formatações de fonte, tamanho e cor.

Depois de digitado e já ter salvo o estilo, retorne ao *Delphi* e no código *.aspx* da sua página, dentro da *tag HEAD*, adicione o código a seguir imediatamente abaixo da *tag* TITLE. Este código a ser adicionado será o responsável por referenciar os estilos recém criados, deixando os mesmos disponíveis para serem utilizados nos componentes da página.

```
<LINK rel="stylesheet"
  href="estilos/estilos.css" type="text/css" />
```

Adicionado a referência ao arquivo de estilos no código *aspx* da página, retorne ao *design* pressionando a tecla F12 e selecione o componente *Calendar Extender,* adicionando em sua propriedade *CssClass* no *Object Inspector* o termo "calendario", nome dado ao estilo no arquivo *estilos.css*. Execute novamente sua aplicação comparando o novo resultado obtido (**Figura 10**) ao primeiro resultado visto na execução anterior, observando que os estilos alteraram a apresentação do componente.

Para aplicarmos também um estilo ao componente *ValidatorCalloutExtender* Criaremos um novo bloco de códigos, o qual terá a finalidade de alterar o design da *popup* que exibe a mensagem dos componentes da *Web Validation*. Retomando o arquivo css *estilos.css* anteriormente criado, adicione ao final do estilo para o calendário o código da **Listagem 3**, encontrando-se também comentado de acordo com a funcionalidade de cada seção de códigos empregados.

Realizada a digitação do código, que se utiliza também de efeitos comuns já conhecidos nos padrões *HTML*, salve as alterações do arquivo e retorne ao formulário da aplicação, onde deverá observar se a propriedade *TargetControlID* do componente *ValidatorCallout* está relacionada ao componente de validação *RegularExpressionValidator*. O mesmo deverá ser selecionado e na janela *Object Inspector* deverá localizar suas propriedades *Extenders* (**Figura 11**). Lembrando que as propriedades apenas estarão visíveis após ambos os componentes estarem interligados, onde as seguintes propriedades deverão ser alteradas conforme itens a seguir:

Figura 10. Resultado do Calendário com Estilo

Nota do DevMan

Cascading Style Sheets, ou simplesmente CSS, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Ao invés de colocar a formatação dentro do documento, o desenvolvedor cria um link (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta portanto modificar apenas um arquivo.

Listagem 3. Estilo css do ValidatorCalloutExtender

```
.validar div, .validar td
/*Formatação do visual da popup(EX: borda e cor de fundo)*/
{border:solid 1px red; background-color:white;}
.validar .ajax__validatorcallout_callout_cell
/*Manter padrão*/
{width:80px; height:100%; text-align:right; vertical-align:top; border:none; background-
color:transparent; padding:0px;}
.validar .ajax__validatorcallout_callout_table
/* Manter padrão */
{height:100%; border:none;background-color:transparent;padding:0px;}
.validar .ajax__validatorcallout_callout_arrow_cell
/* Manter padrão */
{padding:8px 0px 0px 0px; text-align:right; vertical-align:top; font-size:1px;
border:none; background-color:transparent;}
.validar .ajax__validatorcallout_callout_arrow_cell .ajax__validatorcallout_innerdiv
/* Manter padrão */
 {font-size:1px; position:relative; left:1px; border-bottom:none; border-right:none;
border-left:none; width:15px; background-color:transparent; padding:0px;}
.validar .ajax__validatorcallout_callout_arrow_cell .ajax__validatorcallout_innerdiv div
/* Manter padrão */
{height:1px; overflow:hidden; border-top:none; border-bottom:none; border-right:none;
padding:0px; margin-left:auto;}
.validar .ajax__validatorcallout_error_message_cell
/*Formatação do texto dentro da popup */
{font-family:Verdana; font-size:10px; border-right:none; border-left:none; width:100%;
color:red; font-weight:bold; text-align:left;}
.validar .ajax__validatorcallout_icon_cell
/* Manter padrão */
{width:20px; padding:5px; border-right:none;}
.validar .ajax__validatorcallout_close_button_cell
/* Manter padrão */
{vertical-align:top; padding:0px; text-align:right; border-left:none;}
.validar .ajax__validatorcallout_close_button_cell .ajax__validatorcallout_innerdiv
/* Manter padrão */
{border:none; text-align:center; width:16px; height:16px; padding:2px; cursor:pointer;}
```

CloseImageURL: coloque o caminho de onde o componente buscará a imagem para fechar a popup (se não colocar nada, ficará a imagem padrão);

CssClass: digitar o nome do estilo ("validar") criado para sua formatação;

WarningIconImageURL: caminho da imagem que é exibida na frente do texto da *popup*;

Width: Tamanho da *popup* (em caso da mensagem ser muito grande).

O resultado da aplicação de estilos ao componente de validação poderá ser observado na **Figura 12,** onde um formato incorreto de data está sendo ilustrado. Estas variações de estilo você poderá aplicar a seus formulários de acordo com a necessidade e usabilidade dos formulários.

Problemas ocorrem e muitas vezes são inevitáveis, mas sempre há formas de facilitar sua resolução. Nos exemplos vistos durante a criação do artigo, há possibilidades de que possíveis problemas de *javascript* em virtude da compatibilidade dos componentes de *AJAX* possam estar ocorrendo. No *link* a seguir está disponível para download um componente para mapeamento e validação dos códigos, uma ferramenta que será bastante útil para estes casos (*blogs.msdn.com/mattgi/archive/2007/01/23/asp-net-ajax-validators.aspx*).

Realizado o download dos componentes (*Validators.zip*), descompacte-os e localize na pasta uma *dll* chamada *Validattors.dll*. Copie e cole dentro da pasta *Bin* do seu projeto e adicione nas referências da aplicação, onde no *Project Manager* com um clique do botão direito do mouse sobre a *dll* principal do projeto da aplicação, deverá selecionar o item *Add Reference*, buscando na caixa de diálogo da pasta *Bin* do projeto o arquivo *Validattors.ddl* recém adicionado. Para que as configurações e validações entrem em prática de funcionamento, uma alteração no arquivo *Web.Config* deverá ser realizada, adicionando o código da **Listagem 4** dentro da tag *<pages>* localizada entre as demais configurações da declaração *<system.web>* (**Figura 13**).

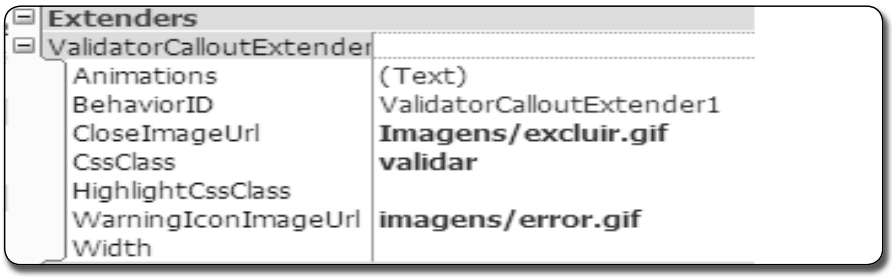


Figura 11. Propriedades Extenders do RegularExpressionValidator

Figura 12. Resultado do ValidatorCalloutExtender com estilo

```
<system.web>
<pages>
    <controls>
     <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions, V
    </controls>
    <tagMapping>
        <add tagType="System.Web.UI.WebControls.CompareValidator"           mappedTagT
        <add tagType="System.Web.UI.WebControls.CustomValidator"            mappedTagT
        <add tagType="System.Web.UI.WebControls.RangeValidator"             mappedTagT
        <add tagType="System.Web.UI.WebControls.RegularExpressionValidator" mappedTagT
        <add tagType="System.Web.UI.WebControls.RequiredFieldValidator"     mappedTagT
        <add tagType="System.Web.UI.WebControls.ValidationSummary"          mappedTagT
   </tagMapping>
</pages>
```

Figura 13. Localização na Web.Config

Conclusão

Vimos no decorrer deste artigo a aplicação de alguns dos componentes disponíveis na tecnologia *AJAX*. A usabilidade destes componentes na sua aplicação torna-a mais leve e dinâmica, facilitando a distribuição dos processos e navegação do usuário. Inúmeros outros recursos poderão ser aplicados com a utilização desta tecnologia, visite o *website* (*www.asp.net/ajax/ajaxcontroltoolkit/samples/*) de apresentação das demais ferramentas e faça a implementação em outras aplicações, criando controles inovadores para suas aplicações. Abraço e até o próximo artigo. ●

Listagem 4. TagMapping

```
<tagMapping>
<add tagType= "System.Web.UI.WebControls.CompareValidator" mappedTagType= "Sample.Web.
  UI.Compatibility.CompareValidator, Validators, Version=1.0.0.0"/>
<add tagType= "System.Web.UI.WebControls.CustomValidator" mappedTagType= "Sample.Web.
  UI.Compatibility.CustomValidator, Validators, Version=1.0.0.0"/>
<add tagType= "System.Web.UI.WebControls.RangeValidator" mappedTagType= "Sample.Web.
  UI.Compatibility.RangeValidator, Validators, Version=1.0.0.0"/>
<add tagType= "System.Web.UI.WebControls.RegularExpressionValidator" mappedTagType=
  "Sample.Web.UI.Compatibility.RegularExpressionValidator, Validators, Version=1.0.0.0"/>
<add tagType= "System.Web.UI.WebControls.RequiredFieldValidator" mappedTagType= "Sample.
  Web.UI.Compatibility.RequiredFieldValidator, Validators, Version=1.0.0.0"/>
<add tagType= "System.Web.UI.WebControls.ValidationSummary" mappedTagType= "Sample.Web.
  UI.Compatibility.ValidationSummary, Validators, Version=1.0.0.0"/>
</tagMapping>
```

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO PROFISSIONAL 1

3 domínios independentes
50 GB de transferência
30 GB de e-mails com SSL
1 GB de espaço
Painel de controle
ASP, ASP.NET 3.5 E PHP 5
Access ilimitado
3 bancos MYSQL 5
1 banco SQL Server 2005 Express

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

Tudo isso por apenas R$ **25,90** por mês

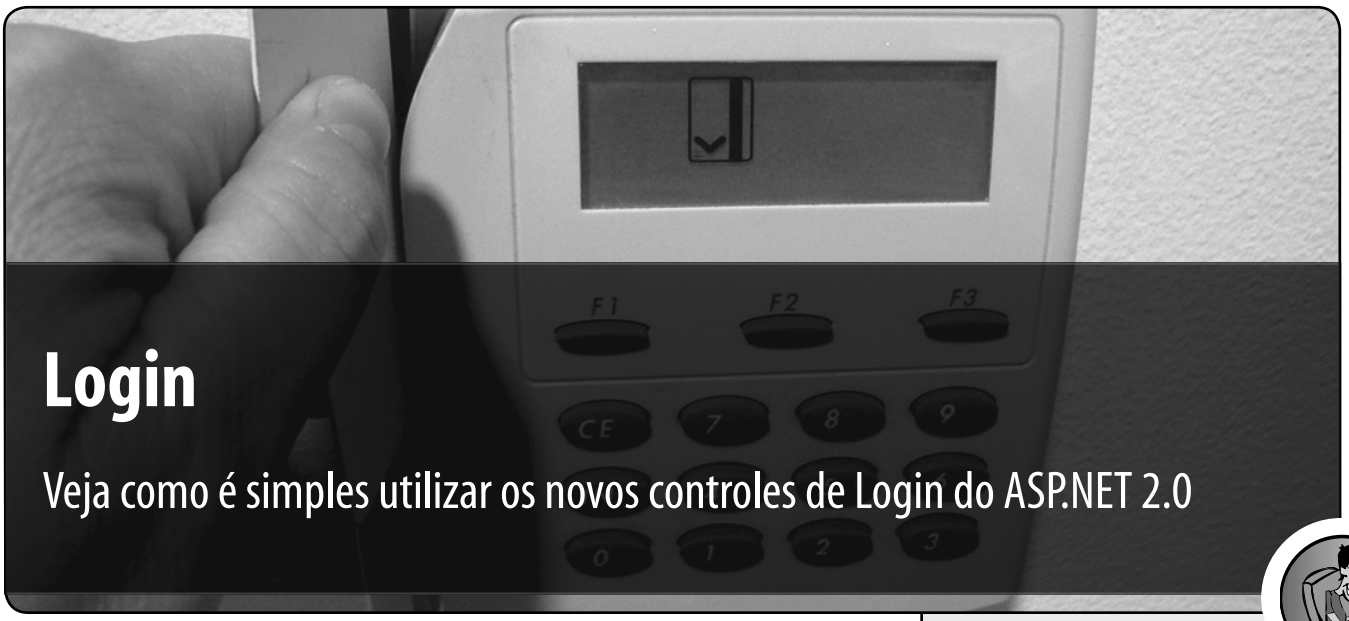
Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET



Login

Veja como é simples utilizar os novos controles de Login do ASP.NET 2.0

Guinther Pauli
(guinther@devmedia.com.br)
é Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine e ClubeDelphi. Desenvolve em ASP.NET há 8 anos.

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi. Gerente de Desenvolvimento da SoftPark, empresa parceira Borland. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi

Nas últimas edições da revista ClubeDelphi vimos alguns artigos voltados à segurança. Vimos artigos mostrando como lidar com usuários no *Firebird* e aplicações Win32 e até dicas de segurança no PHP. O que veremos nesse artigo tem tudo a ver com segurança. O *.NET Framework 2.0* está repleto de funcionalidades que visam agilizar o trabalho do desenvolvedor, incluindo controles de *login*. Um exemplo claro disso são os controles de *Login* do *ASP.NET 2.0*.

Aliando o uso dos controles de *Login* com o novo esquema de segurança do *ASP.NET 2.0*, podemos implementar segurança em nossas aplicações sem uma linha de código sequer, como veremos neste artigo. Isso mesmo! Com o *ASP.NET 2.0* podemos criar telas para incluir usuários, solicitar *login* ao acessar determinada página, modificar senha etc. O mais interessante disso é que não precisaremos programar praticamente nada em nossa aplicação.

Resumo DevMan

Os novos controles de Login do ASP.NET 2.0 facilitam e muito a realização de tarefas rotineiras no que diz respeito à segurança de um Web Site: criação de usuários de grupos, regras de acesso, troca de senha, lembrar senha, visualização de parte do site de acordo com o grupo e muito mais.

Nesse artigo veremos

- Utilização dos controles de Login;
- Integração do aspnetdb com o banco de dados da aplicação.

Qual a finalidade

- Demonstrar a facilidade no uso de controles de login do ASP.NET 2.0.

Quais situações utilizam esses recursos?

- Em praticamente todos os sites que necessitem de autenticação.

Links

Neste artigo, utilizaremos o SQL Server 2005 Express, free, comprovadamente o MELHOR banco para se trabalhar com aplicações ASP.NET.

SQL Server 2005 Express
http://www.microsoft.com/express/download/

Controle de Login

Para iniciarmos nosso estudo, vamos criar uma pequena aplicação capaz de demonstrar todos os recursos de *login* no *ASP.NET 2.0* utilizando o *RAD Studio 2007*. Crie uma nova aplicação usando o menu *File>New>ASP.NET Web Application – Delphi .NET*. Em *Name* digite *Login*, que será o nome de nossa aplicação, e em *Location* indique o local onde os fontes do sistema serão salvos. Mantenha o padrão (*C:\Inetpub\wwwroot*), pois o diretório *\Inetpub\wwwroot* do Windows já possui todas os detalhes de acesso e segurança configurados no *IIS*(*Internet Information Server*). Troque apenas o nome da pasta do projeto se preferir. Mantenha o servidor de testes como IIS e confirme (**Figura 1**).

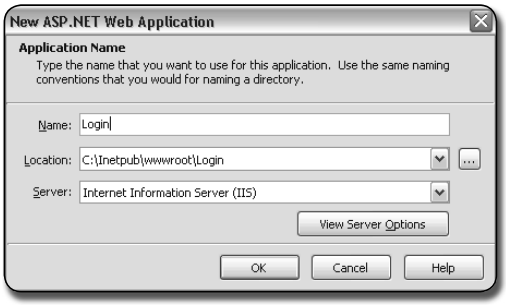


Figura 1. Criação do projeto de login

Figura 2. Controles de Login do ASP.NET 2.0

A primeira coisa que vamos observar é a paleta de componentes *Web Login* (**Figura 2**). Como podemos ver, temos uma série de controles específicos para criação de *login* em nossos sistemas Web.

Arraste o controle *Login* para sua página *Default.aspx*. Veja que temos todos os elementos de um componente que efetua *logins* de um site. Como qualquer controle *ASP.NET*, podemos alterar o formato de *layout* do controle acessando a opção *Auto Format* que encontramos em *Tasks*. Em nosso exemplo foi aplicado o formato *Professional*. Você pode conferir o resultado na **Figura 3**.

Adicione um novo *Web Form* ao projeto chamado *WebMenu.aspx*. Essa será a página que chamaremos caso o usuário informe um usuário e senha válidos. Para isso acesse o menu *File>New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* e *design* do *Menu.aspx*, inclua o texto *Olá, você está logado!*. Antes da vírgula inclua o controle *LoginName*.

Volte ao *Default.aspx*, clique no *Login* e vá em propriedades. Procure a propriedade *DestinationPageUrl*, clique no botão com três pontos e escolha a página *WebMenu.aspx*. Execute seu projeto, informe em *UserName* o usuário da sua máquina local e sua respectiva senha. Veja o resultado na **Figura 4**.

Esse erro ocorre porque ainda não configuramos nada no *ASP.NET Configuration Site*, que já veremos a seguir. Antes de mais nada, é possível traduzirmos as mensagens que aparecem nas telas de *login*. Basta acessar a propriedade *FailureText* do componente e efetuar a mudança. Para traduzir os rótulos das caixas de texto, caixas de opções e botões acesse as propriedades: *UserNameLabelText*, *PasswordLabelText*, *RememberMeText* e *LoginButtonText*, respectivamente.

Figura 3. Controle Login na Default.aspx

Figura 4. Falha ao tentar logar no site

ASP.NET Configuration

Através do *ASP.NET Configuration Site* podemos trabalhar toda a segurança da aplicação. Temos condições de criar usuários, adicionar permissões ou negar acesso a determinadas páginas, incluir grupos de regras (*roles*) e uma série de outros recursos de segurança.

Vamos entender agora como funciona o novo esquema de segurança do *ASP.NET 2.0*. Vá até o *Project Manager*, clique com o botão direito e escolha a opção *ASP.NET Configuration Site*. O *browser* será aberto e nele uma tela como a **Figura 5** será exibida.

Clique na guia *Security*, onde a primeira configuração necessária é a autenticação. Utilizaremos o *wizard* para nos auxiliar no processo de configuração. Clique em *Use the security Setup Wizard to configure security step by step*. Clique em *Next*, pois a primeira tela que visualizamos é apenas um resumo das tarefas que serão realizadas durante o assistente. Logo em seguida configuraremos o modo como nossa aplicação fará a autenticação. Temos duas opções:

- *Internet (Forms)*: a autenticação é baseada em um formulário de *login* e a autenticação feita "na mão" pelo desenvolvedor;
- *Intranet (Windows)*: a autenticação é baseada no *login* na rede, ou seja, não necessita de nenhum trabalho de auten-

Nota

Quando criamos uma aplicação Web no RAD Studio 2007, ele nos cria o arquivo Web.Config totalmente pré-configurado para acessar o banco de dados BlackFish da própria CodeGear. Portanto, ao iniciar uma nova aplicação no RAD teremos que modificar esse arquivo para que não ocorram erros inesperados ao executar o Web Site. Vejamos como fazer isso.

Acesse o arquivo Web.Config no Project Manager e localize, primeiramente, a *tag* <*connectionStrings*>. Remova todo o conteúdo dela deixando-a apenas como mostrado a seguir:

```
<connectionStrings>
</connectionStrings>
```

Logo abaixo de <*connectionStrings*> encontraremos outra *tag* que precisa ser removida, <*system.data*>. Limpe-a também deixe-a como a seguir:

```
<system.data>
</system.data>
```

Em seguida, ao final do arquivo, localize as *tags* <*profile*> e <*roleManager*>. No conteúdo dessas *tags* você encontrará um atributo chamado *connectionStringName*. Troque seu nome para *LoginDevMedia* em ambas.

Pronto! Com isso podemos rodar normalmente nossa aplicação.

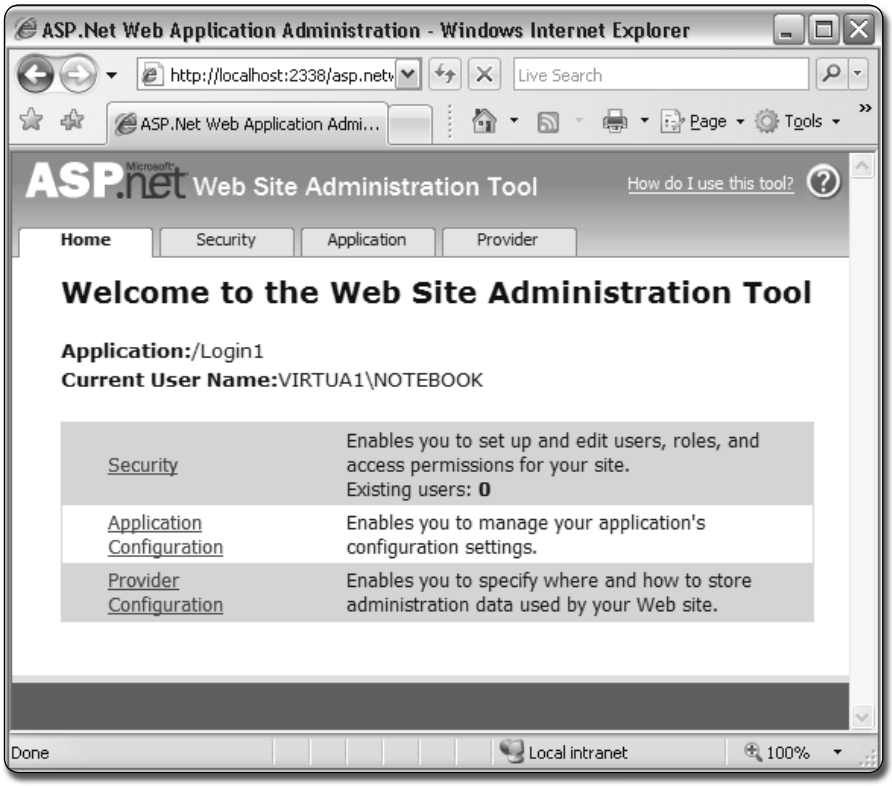


Figura 5. Página inicial do ASP.NET Configuration Site

Nota do DevMan

Por padrão os componentes da paleta *WebLogin* possuem seus *Label's* em inglês. Isso pode ser facilmente modificado, pois há diversas propriedades que podemos modificar e traduzir para a linguagem local. Vejamos o que podemos modificar para que o usuário não se perca na hora de efetuar o *login* ou mesmo cadastrar-se. Pressione *F11* com o componente *LoginView* selecionado e veja suas propriedades:

- *UserNameLabelText:* Troque de *User Name:* para "Nome de Usuário:";
- *PasswordLabelText* Modifique para "Senha:";
- *PasswordRequiredErrorMessage:* "Senha obrigatória";
- *UserRequiredErrorMessage:* "Nome de Usuário obrigatório";
- *RememberMeText:* "Lembrar meu usuário";

Note que, ao modificar a propriedade, o *template* é automaticamente atualizado na *IDE* do *RAD Studio 2007*. Além disso, perceba que outras propriedades podem sofrer modificações, tais como: *FailureText,* texto apresentado quando alguma falha de *login* ocorre e LoginButtonText, ou seja, o texto do do botão *Login*.

A seguir veremos a inclusão do controle *CreateUserWizard* que também pode ser altamente modificado pra a linguagem local. Você contará com algumas propriedades importantes, como por exemplo:

- *UserNameLabelText:* Texto para o *Label* do *TextBox* de nome de usuário;
- *UserNameRequiredErrorMessage:* Mensagem apresentada ao requerer nome de usuário;
- *UnknownErrorMessage:* Mensagem de erro desconhecido;
- *QuestionLabelText:Label* para o campo de pergunta secreta;
- *QuestionRequiredErrorMesasge:* Mensagem apresentada ao requerer nome de usuário;
- *PasswordLabelText:* Texto para o *Label* do *TextBox* de senha;
- *PassowordRequiredErrorMessage:* Mensagem de texto ao requerer senha;

E ainda o textos dos botões:

- *ContinueButtonText:* Texto do botão Continue ("Continuar");
- *CreateButtonText:* Texto do botão *Create User* ("Confirmar criação de usuário");
- *FinishButtonText:* Texto do botão *Finish* ("Finalizar");
- *StartNextButtonText:* Texto do botão Next ("Próximo passo");

Entre outras mensagens e textos que podem ser modificados. Em resumo, os controles do *ASP.NET 2.0* podem ser personalizados em praticamente tudo, facilitando a vida do desenvolvedor e agilizando o processo de criação e manutenção do site.

ticação. É mais utilizado em aplicações corporativas.

Vale lembrar que muitas configurações de segurança usarão um *BD* especial criado automaticamente pelo *ASP.NET*, o *aspnetdb*. Novamente clique em *Next*. Faremos agora algumas configurações avançadas. Habilite a opção *Enable roles for this Web Site*, dessa forma estamos prevendo que nossa aplicação será capaz de validar grupos de usuários para ter acesso a determinadas partes do site.

> **Nota**: Não precisamos alterar nada em relação ao armazenamento, vamos utilizar o padrão do *ASP.NET*. O *ASP.NET* cria automaticamente um banco no *SQL Server Express*, chamado *aspnetdb*, que armazena as informações de configuração das aplicações criadas, inclusive todas as informações relacionadas à segurança.

Prosseguindo com o *wizard*, nessa etapa temos que definir se nosso *Web Site* trabalhará com *Roles,* como já foi mencionado. Clicando novamente em *Next,* o assistente solicitará o nome do grupo para que ele possa criar para nós. Os usuários são agrupados em *Roles* para identificar que eles desempenham as mesmas funções na empresa, como por exemplo: Vendas, *Marketing*, Administradores etc. Use *Roles* em aplicações com muitos usuários, pois você pode aplicar regras de segurança diretamente a esses grupos e então incluir usuários a elas. Isso facilita a manutenção dos usuários da aplicação. Para criar uma *Role* basta informar o seu nome no campo *New Role Name* e clicar no botão *Add Role*. Para o nosso exemplo vamos criar algumas *Roles*: *Vendas, Marketing, Administradores* e *Usuarios*. Veja na **Figura 6** que as *Roles* criadas aparecem em uma lista, onde podemos inclusive excluí-las, através do link *Delete*. Após criar as *Roles* clique em *Next*.

Criando usuários

Agora temos a opção de criar usuários. Vamos criar alguns usuários para podermos realizar nossos testes. Podemos apenas informar o nome do usuário, senha, e-mail e adicionalmente uma pergunta e resposta para auxiliar a lembrar da senha em caso de esquecimento. Ao

terminar o preenchimento dos campos clique em *Create User*. Você pode criar dois usuários com os nomes e demais informações da sua escolha (**Figura 7**). Após criar os dois usuários necessários para nosso exemplo, clique em *Next*.

> **Nota:** Por padrão, o *ASP.NET* usa uma política de segurança para as senhas, que devem ser maiores que sete dígitos e requer um caractere especial (*, &, % etc.).

Access Rules

Nessa última etapa do *wizard* podemos definir as *Access Rules*. Não confunda *Role* com *Rule*. As *Access Rules* são regras de acesso, que podem servir para permitir ou negar o acesso a determinadas áreas do site. Você pode utilizar as *Access Rules* para controlar o acesso de toda a aplicação ou de cada um das *Folders* (pastas) da aplicação. As regras de acesso podem ser aplicadas a um único usuário, a uma *Role*, a todos os usuários ou somente a usuários anônimos.

Para adicionar uma *Access Rule* selecione primeiramente a pasta na qual ela será aplicada. Se a *Access Rule* for válida para toda a aplicação, selecione o *Folder* raiz. Em seguida defina se a *Access Rule* será aplicada a uma *Role*, usuário, todos os usuários ou somente os usuários anônimos. Para finalizar você precisa definir se a *Access Rule* irá garantir o acesso à pasta ou negá-lo. Após definir esses três pontos, clique em *Add This Rule*.

As *Rules* adicionadas irão aparecer em uma lista logo abaixo. As regras de acesso são aplicadas na ordem dessa lista, e vale a primeira que for encontrada, por exemplo: se a primeira *Rule* nega o acesso aos usuários anônimos na aplicação, e a segunda garante, fica valendo a primeira. Nessa lista podemos excluir as *Rules* através do link *Delete*.

Em nosso exemplo iremos apenas criar duas *Access Rules* simples. Uma garantirá acesso aos usuários da *Role Vendas*, e a outra que irá garantir o acesso aos usuários da *Role Marketing* (**Figura 8**).

Após a criação das *Access Rules*, clique em *Next*. Uma tela de conclusão irá aparecer, basta agora clicar em *Finish*. Você pode gerenciar usuários, *Roles* e *Access*

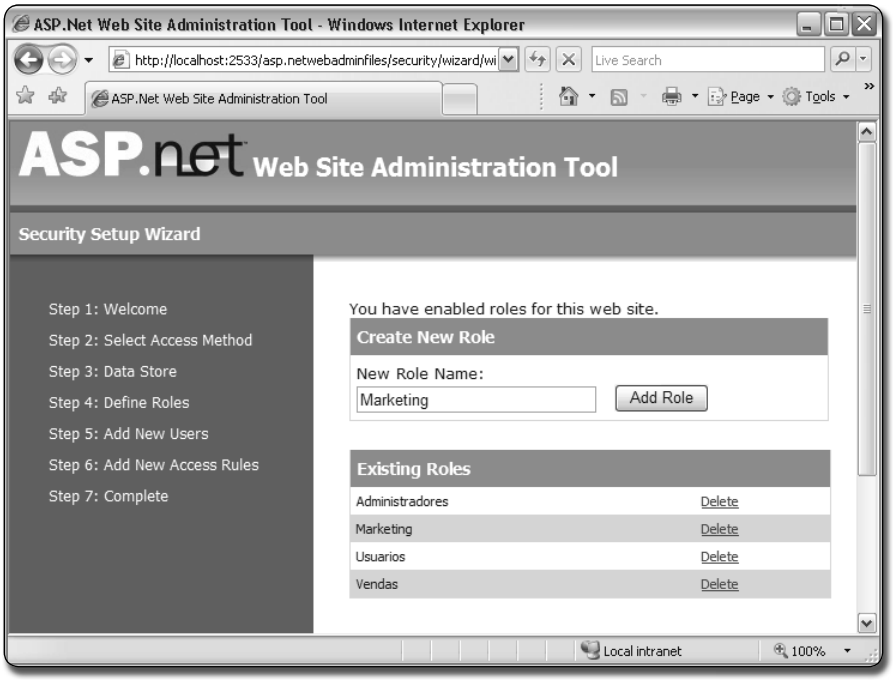


Figura 6. Criando Roles

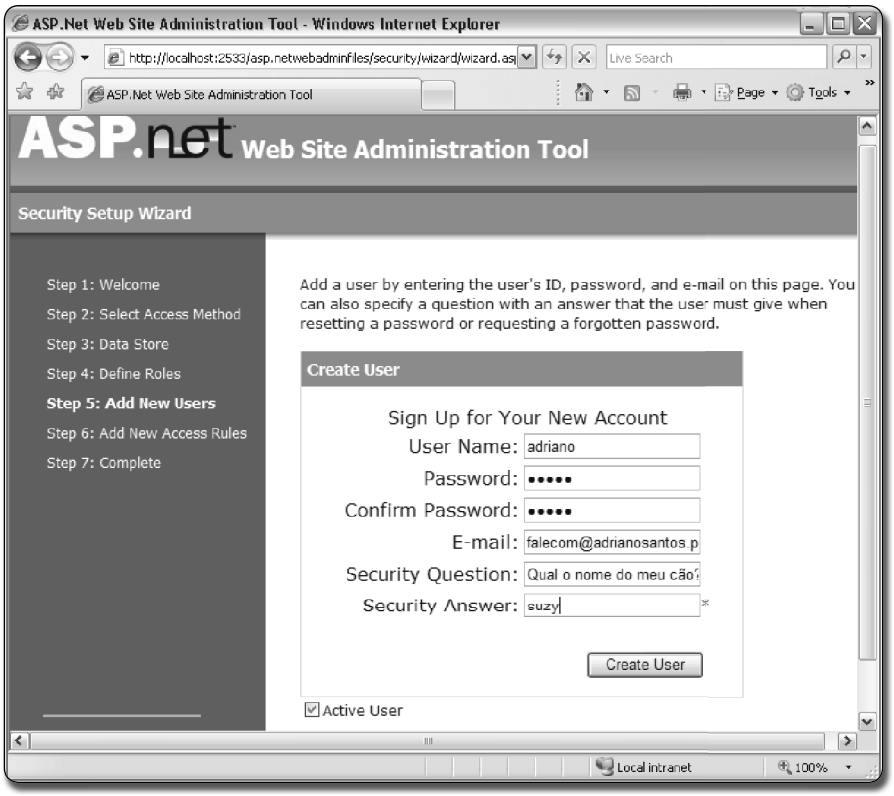


Figura 7. Criando um novo usuário no ASP.NET 2.0

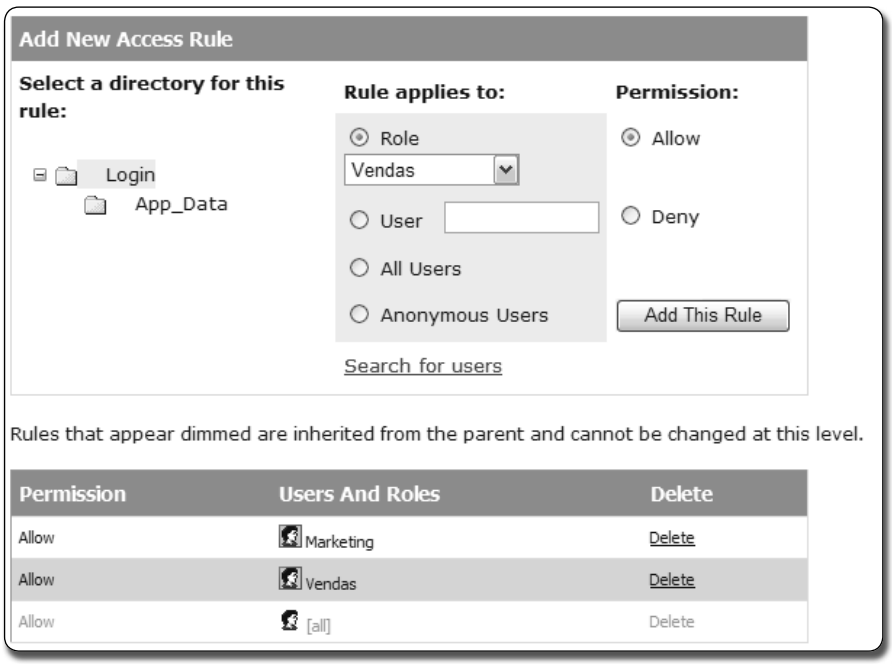


Figura 8. Definindo Access Rules

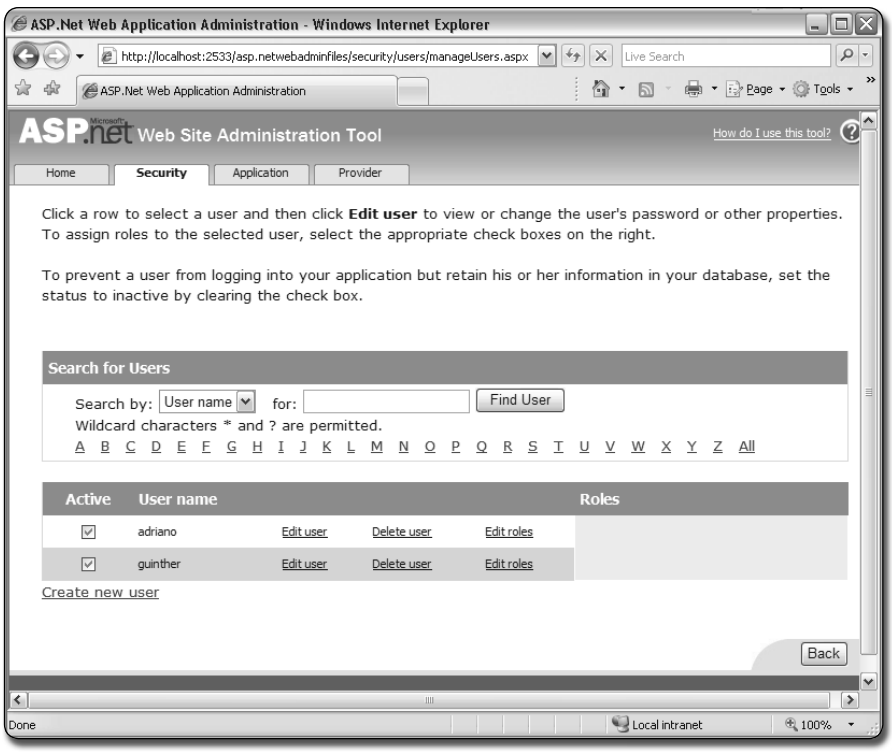


Figura 9. Definindo Roles dos usuários

Rules no momento que quiser, basta acessar o *ASP.NET Configuration* e ir até a aba *Security*

Atribuindo usuários às roles

Você deve ter percebido que está faltando algo em nossa configuração. Ainda não definimos qual é a *Role* de cada um dos usuários que criamos. É exatamente isso que faremos agora. Na aba *Security*, do *ASP.NET Configuration*, clique na opção *Manage Users*. Uma tela como a da **Figura 9** aparecerá.

Para definir quais são as *Role* de um usuário, você deve clicar no link *Edit roles*, que fica ao lado de cada usuário da lista de usuários. Ao clicar no link, à direita nos aparece uma lista com todas as *Roles* disponíveis. Basta selecionar as *Roles* do usuário em questão. Em nosso exemplo, vamos definir que o usuário *adriano* participa da *Role Vendas*, e que o usuário *guinther* participa da *Role Marketing*

Testando o acesso do usuário

Com certeza, agora nosso sistema já funcionará perfeitamente após as configurações no *ASP.NET Configuration Site*, porém para termos certeza disso vamos fazer um pequeno teste. Volte ao *RAD Studio 2007*, compile e execute a aplicação. No controle de *Login*, informe o usuário *adriano ou guinther* e sua respectiva senha. Veja o resultado na **Figura 10**.

Controle LoginView

O controle *LoginView* é um dos controles mais interessantes do *ASP.NET 2.0*. Com ele podemos determinar quais áreas cada grupo de usuários tem acesso e conseqüentemente quais controles de tela aparecerão. Vamos então utilizar o controle *LoginView* para validarmos a utilização das *Roles*. O *LoginView* é um controle utilizado para definir diferentes conteúdos em determinadas áreas de uma página, de acordo com a *Role* do usuário, assim como já mencionado. Abra a página *WebMenu.aspx*, e inclua um controle *LoginView*. Nas *Tasks* do controle clique na opção *Edit RoleGroups*. Aparecerá uma janela como demonstrado na **Figura 11**.

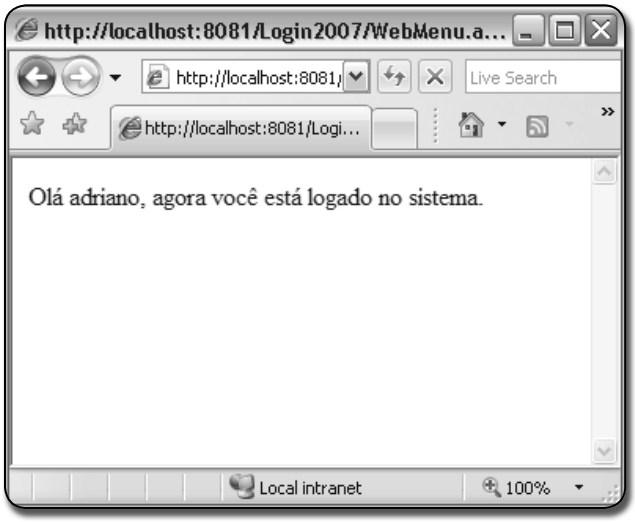


Figura 10. Usuário autenticado na aplicação

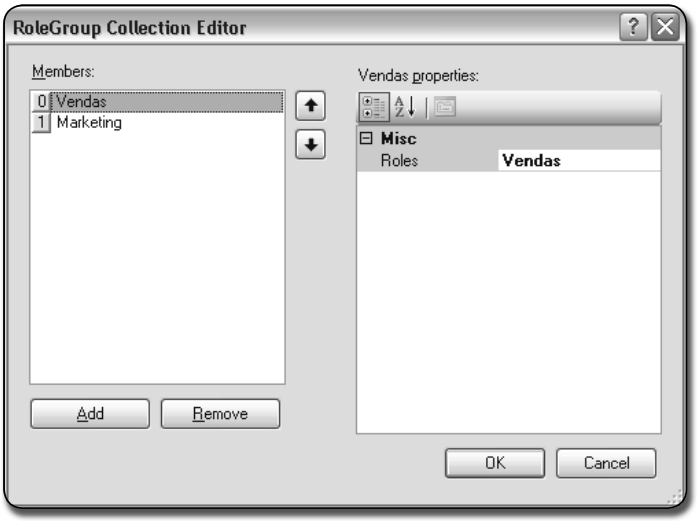


Figura 11. Configurando RoleGroups no LoginView

Clique em *Add* e aparecerá um item chamado *RoleGroup*. À direita teremos uma propriedade chamada *Roles*, clique no botão com três pontos referente a essa propriedade. Aparecerá uma janela para informarmos o nome da *Role*. Digite *Vendas* e clique em *OK*. Clique novamente em *Add* e repita esse procedimento para a *Role Marketing*. Ao finalizar clique em *OK*.

Ainda em *Tasks*, observe a opção *Views*. Como mostra a **Figura 12**, podemos escolher diversas *Views* possíveis e definir um conteúdo diferente para cada uma delas.

Vamos selecionar a *View* de *Vendas*. Veja que um *LoginView* funciona como um *container*, onde podemos inserir controles dentro dele. Vamos apenas inserir o texto *Esse é o conteúdo para quem é de Vendas*. Modifique para a *View* do *Marketing* e insira o texto *Esse é o conteúdo para quem é de Marketing*.

Feitas essas alterações, salve, compile e execute. Acesse primeiramente, utilizando o usuário *adriano*. Veja que ele tem acesso apenas ao conteúdo da *Role*

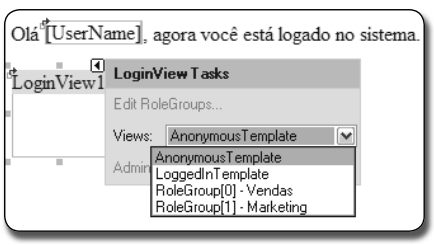
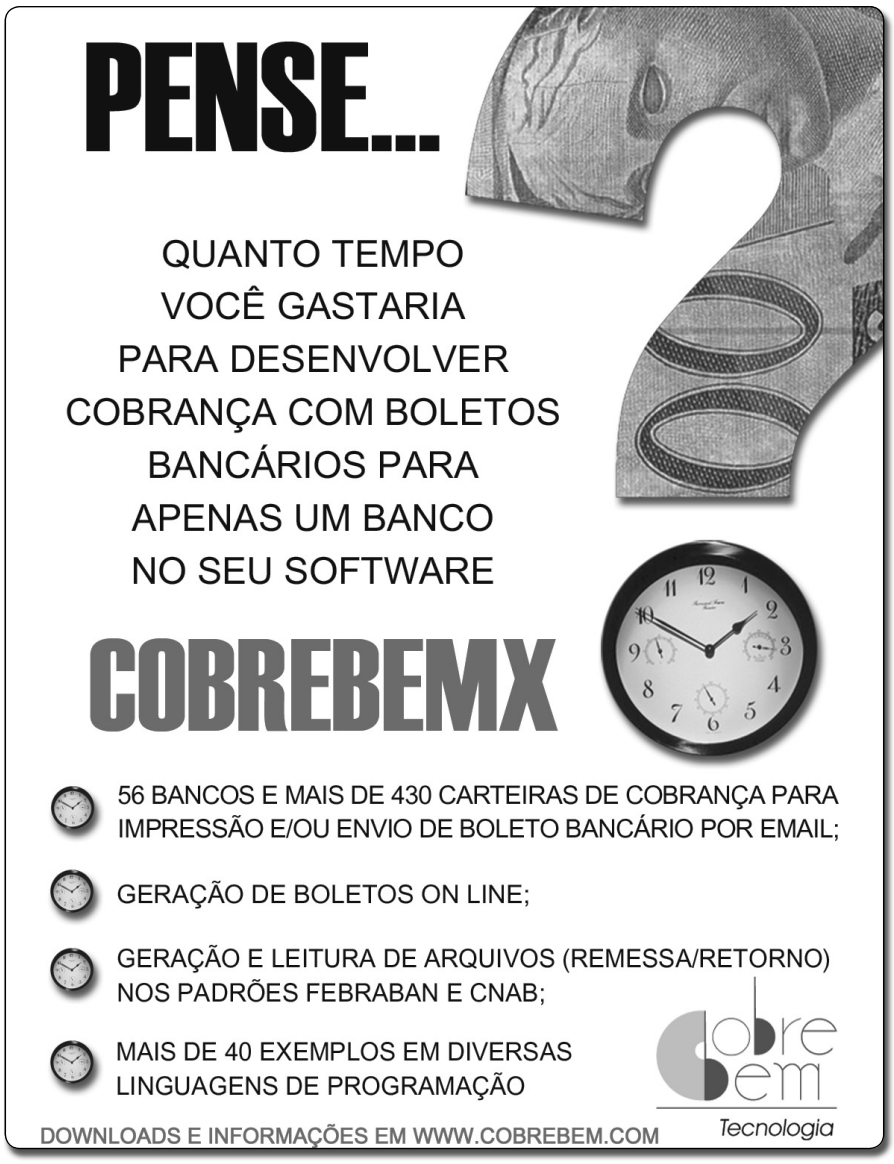


Figura 12. Possíveis Views do controle LoginView



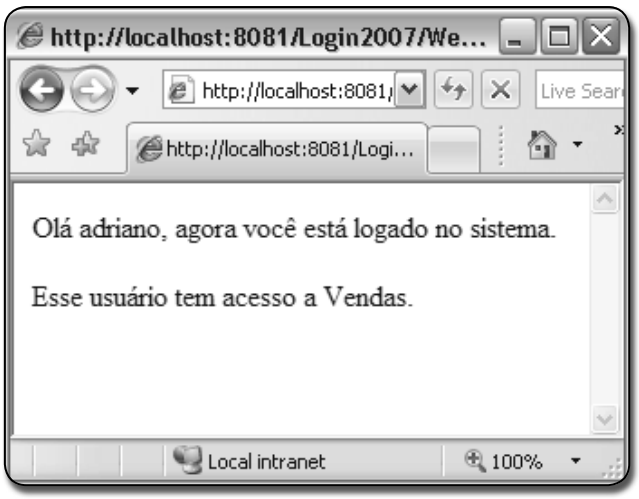


Figura 13. Usuário do grupo Vendas

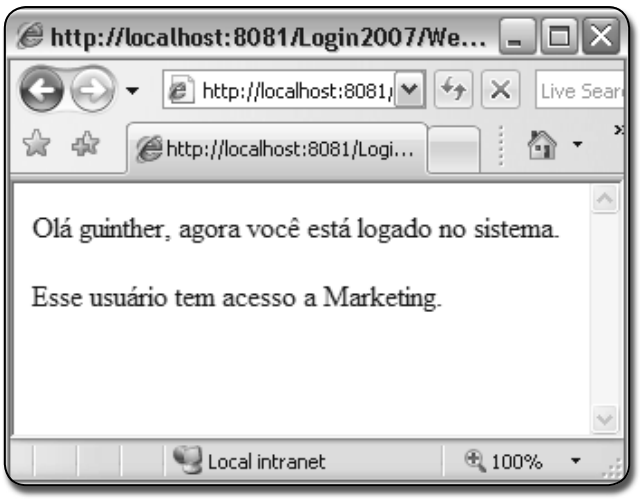
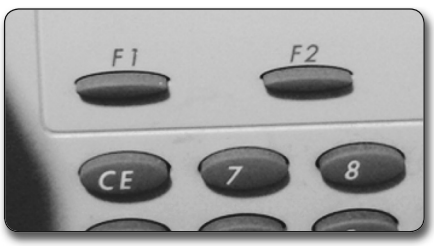


Figura 14. Usuário do grupo Marketing

Figura 15. Controle de Login com opção de cadastrar novo usuário

Figura 16. Modificando a senha com o controle ChangePassword



de *Vendas*. Saia e acesse com o usuário *guinther*. Veja que ele apenas acessa o conteúdo da *Role* de *Marketing*. É possível também definir um conteúdo diferente para usuários anônimos, ou simplesmente para usuários *logados* que não pertençam a nenhuma *Role* configurada (**Figuras 13** e **14**).

Controle CreateUserWizard

Vamos utilizar agora um controle *CreateUserWizard*. Adicione um novo *WebForm* ao projeto chamado *CreateUser.aspx*. No *design*, insira um *CreateUserWizard*. Configure o *AutoFormat* para *Professional*. Nas propriedades do controle e na propriedade *ContinueDestinationPageUrl* informe a página *Default.aspx*, para indicar a página de destino após a criação da nova conta.

Volte na página *Default.aspx* e vá até as propriedades do controle de *Login*. Localize a propriedade *CreateUserText* e digite *Novo Usuário*. Logo abaixo, na propriedade *CreateUserUrl*, informe a página que acabamos de criar, usando o botão de reticências. Veja na **Figura 15** como fica nosso formulário de *login*. Salve, compile e execute o projeto. Perceba que ao clicar em *Novo usuário* o formulário para criação de usuário é chamado e podemos nos cadastrar normalmente no site agora.

Clique no link *Novo usuário* e veja que podemos incluir um novo usuário, assim como fizemos pelo *ASP.NET Configuration*. Preencha o formulário para a criação do novo usuário e clique em *Create User*. Irá aparecer uma mensagem informando que o usuário foi criado com sucesso, clique no botão *Continue*. Você voltará à tela de *login*, informe o usuário e senha que você acabou de criar e clique em *Login*. Pronto, você já tem um novo usuário habilitado!

Controle ChangePassword

Temos ainda a possibilidade de permitir que a senha seja alterada usando o controle *ChangePassword*. Com esse controle o usuário modifica sua própria senha sem a necessidade de entrar em contato com o administrador do site. Vamos ao exemplo: apenas acione o *design* da página *WebMenu.aspx*. Inclua um controle do tipo *ChangePassword* e configure o seu formato para *Professional* clicando no link *Auto Format*. Salve, compile e execute. Informe o usuário e senha no controle de *Login*. Veja que em nossa página *WebMenu.aspx* já temos disponível um controle para modificação de senhas (**Figura 16**). Faça um teste e modifique a senha do usuário.

Integrando a segurança do ASP.NET em seu banco

Certamente você deve estar se perguntando como integrar esse esquema de segurança em seu próprio banco de dados, uma vez que o *ASP.NET* armazena seus usuários no *aspnetdb*. Nos exemplos que fizemos até agora, utilizamos o *aspnetdb*, que foi automaticamente criado no *SQL*

Server Express. Esse é um tipo de solução possível, mas também é possível integrar todo o esquema de segurança do *ASP.NET* em nosso próprio banco.

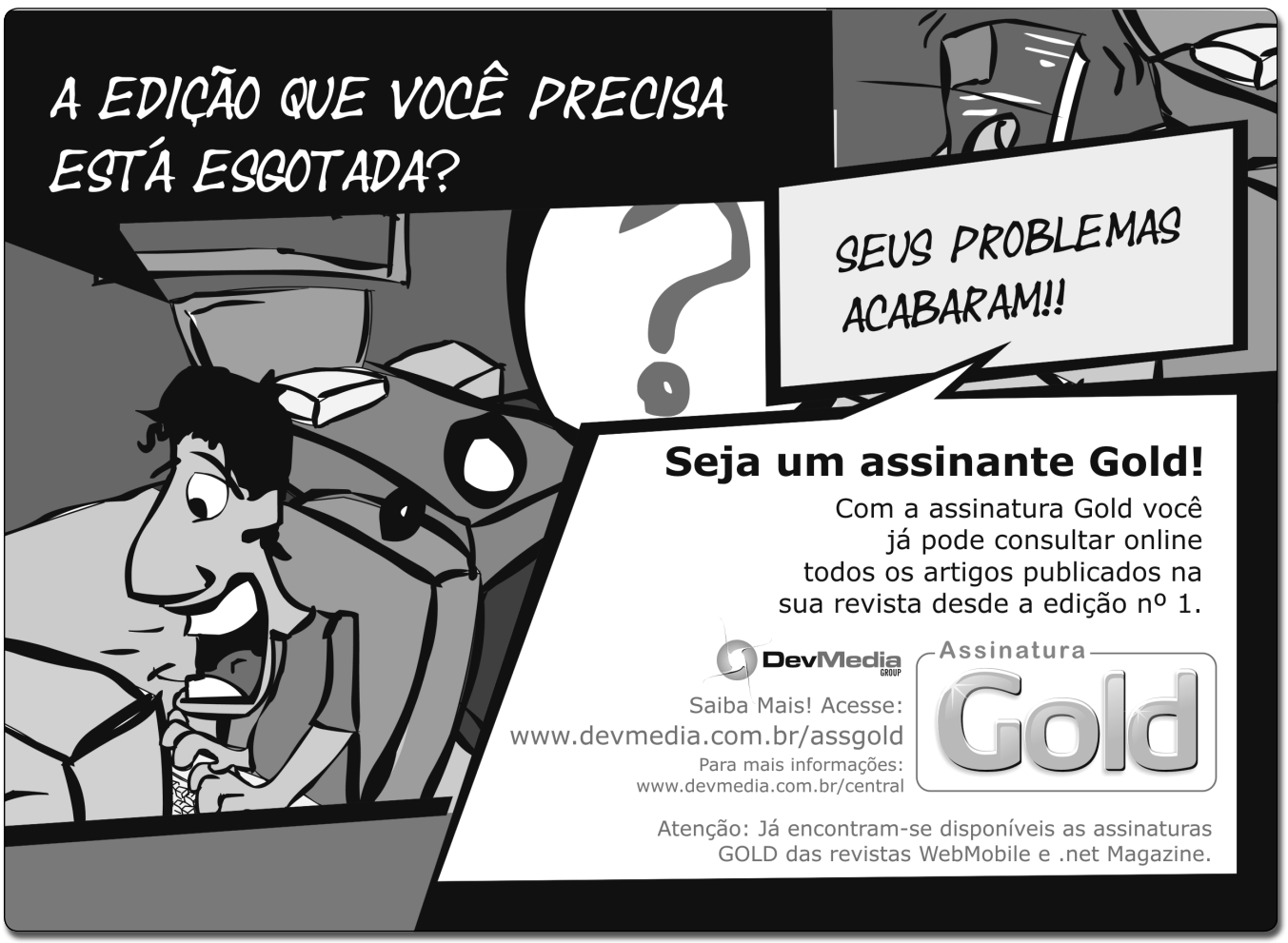
Quando o *ASP.NET 2.0* é instalado, temos um aplicativo chamado *aspnet_regsql.exe*, que fica no diretório *C:\Windows\Microsoft.NET\Framework\v2.0.50727*. Através desse aplicativo podemos criar toda a infra-estrutura de tabelas de segurança em nosso próprio banco do *SQL Server*. Essa estrutura de que falamos é a que foi demonstrada anteriormente na **Figura 7**.

Vamos fazer um teste criando essa estrutura em um banco diferente, nesse caso em nosso próprio banco.

> **Nota** : O *download*, instalação e criação do banco de dados não serão abordados aqui, pois fogem do escopo desse artigo.

Pelo *prompt* de comando vá até o diretório indicado e execute o aplicativo

Figura 17. Definindo o banco Northwind para criação da estrutura do Application Services



aspnet_regsql.exe. Um *wizard* será iniciado. Clique em *Next* para prosseguir. Na próxima tela o *wizard* pergunta se queremos criar a estrutura do *Application Services* ou se queremos removê-la. Vamos selecionar a primeira opção e clicar em *Next*.

> **Nota** : É importante salientar que a estrutura de tabelas criada pelo *ASP.NET* não se destina apenas à segurança, e sim a um conjunto de serviços (*Application Services*) que podemos implementar em nossas aplicações.

Na próxima janela devemos definir o servidor e o banco *SQL Server* onde criaremos a estrutura, como mostra a **Figura 17**. Veja que definimos o banco *Northwind*, um banco padrão do *SQL Server* usado para testes, semelhante ao *Employee* do *Firebird*. Em seguida clique em *Next*. A próxima janela é apenas uma confirmação, clique em *Next* novamente e aguarde alguns instantes enquanto a estrutura é criada. Em seguida, basta clicar em *Finish*. As tabelas do *ASP.NET* foram agora criadas no banco *Northwind*.

O *ASP.NET* acessa e persiste as informações do *Application Services* através de classes chamadas de *Providers***.** Precisamos agora configurar nossa aplicação para que o *provider* responsável pelo acesso aos dados de segurança da nossa aplicação acesse o banco *Northwind* e não o *aspnetdb*. Você deve configurar o seu arquivo *web.config* como mostra a **Listagem 1**.

Observe que temos uma *ConnectionString* que aponta para o *Northwind*, e logo abaixo temos a configuração do *provider AspNetSqlMembershipProvider* e do *provider AspNetSqlRoleProvider*. Veja que estamos apontando para a *ConnectionString* criada, e que também podemos definir uma série de configurações de segurança aqui, como por exemplo, o tamanho mínimo das senhas, no atributo *minRequiredPasswordLenght*. Pronto, nossa aplicação e banco já têm a segurança do *ASP.NET* implementada! Basta agora criar os usuários, *Roles* e *Access Rules* necessários.

Caso queira, também pode criar relacionamentos entre as tabelas do nosso sistema com a tabela *aspnet_Membership*. Dessa forma, como exemplo, podemos identificar que o usuário logado é determinado funcionário da tabela *Employees* ou determinado cliente da tabela *Customers*, e aí aplicar as regras de negócio da nossa aplicação. Tudo depende da necessidade e realidade de cada solução.

> **Nota:** Adicionalmente, é possível customizar os *providers* de *Membership* e *Roles*, para que acessem uma estrutura de tabelas diferente da criada e utilizada pelo *ASP.NET*. Isso é útil caso nosso banco não seja um *SQL Server*, ou caso queiramos aproveitar uma tabela de usuários já existente.

Listagem 1. Web.config alterado para configurar Provider e ConnectionString

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
<connectionStrings>
  <clear/>
  <add
    name="SqlConnectionString"
  connectionString="Data Source=SQLSERVER2005;Initial Catalog=NorthWind;Integrated
Security=True;"/>
</connectionStrings>

<system.web>
<authentication mode="Forms" />
  <membership defaultProvider="AspNetSqlMembershipProvider">
  <providers>
<clear/>

<add
  name="AspNetSqlMembershipProvider"
  type="System.Web.Security.SqlMembershipProvider"
  connectionStringName="SqlConnectionString"
  applicationName="NorthWind"
  enablePasswordRetrieval="false"
  enablePasswordReset="true"
  requiresQuestionAndAnswer="false"
  requiresUniqueEmail="false"
  passwordFormat="Hashed"
  minRequiredNonalphanumericCharacters="0"
  minRequiredPasswordLength="8"/>
  </providers>
  </membership>

<roleManager enabled="true"
  defaultProvider="AspNetSqlRoleProvider">
    <providers>
    <clear/>
    <add
      name="AspNetSqlRoleProvider"
      type="System.Web.Security.SqlRoleProvider"
      connectionStringName="SqlConnectionString"/>
  </providers>
    </roleManager>
</system.web>
</configuration>
```

Conclusão

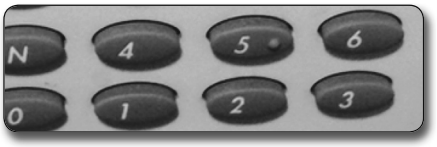
O *ASP.NET 2.0* veio repleto de novidades em segurança. Pudemos ver que juntamente com os controles de *Login*, é possível implementar segurança em nossas aplicações sem precisar incluir uma única linha de código! Fizemos tudo apenas configurando a segurança pelo *ASP.NET Configuration* e através dos controles de *Login*. Um abraço a todos e até a próxima! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



local
Data Center

Out Sourcing Total

Preparados para nova era em qualidade de serviços, garantimos operações full-time dentro de uma gestão única desenvolvida pela nossa equipe para atuar preventivamente, somos a única empresa a operar por 25hs por dia garantindo disponibilidade total.

Escalabilidade, Conectividade e Disponibilidade.

- Servidor Dedicado
- Gerenciamento
- Banda IP
- Cross Connection
- Link PAP
- Serviços compartilhados
- Sites
- E-mail
- Backup

CERTIFICAÇÕES

- MICROSOFT
- LINUX
- CISCO
- ITIL

Data Center Próprio

Segurança e certeza
de bons negócios e serviços

contato@localdatacenter.com.br

+55(21) 3527-6510

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Navegação

Trabalhando com Menus, SiteMap, MasterPages e TreeView no ASP.NET 2.0

Guinther Pauli
(guinther@devmedia.com.br)
é Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine e ClubeDelphi. Desenvolve em ASP.NET há 8 anos.

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi. Gerente de Desenvolvimento da SoftPark, empresa parceira Borland. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi

Resumo DevMan

O ASP.NET 2.0 trouxe novos controles que facilitam a criação da estrutura de navegação de um Web Site. Um SiteMap permite definir toda a estrutura do site em XML. Controles de navegação como menu e treeview se ligam ao mapa do site para prover a interface. Master Pages auxiliam na criação do Web Site por permitirem a definição de um layout comum.

Nesse artigo veremos

- Utilização de Master Page;
- Definição do mapa do site usando XML;
- Usando Treview, Menu e SiteMapPath.

Qual a finalidade

- Demonstrar a facilidade no uso de controles de menu do ASP.NET 2.0.

Quais situações utilizam esses recursos?

- Em praticamente todos os sites que necessitem navegação.

Interfaces de navegação de sites sempre foram uma questão complicada. O que usar para criar uma interface amigável ao usuário, e ao mesmo tempo simples de fazer manutenção? O *ASP.NET 2.0* trouxe recursos que tornaram essa tarefa tão simples que parece brincadeira. Este artigo irá explicar como *Master Pages*, *Site Map*, *Menus* e *TreeView* podem nos ajudar a criar estruturas de navegação sem uma única linha de código!

Master Pages

As *Master Pages* são páginas mestre, como o próprio nome sugere. Elas substituem os antigos *User Controls* do *Framework 1.x*. Uma vez criadas, as *Master Pages* podem ser incorporadas a qualquer página do *Web Site* e, por serem únicas, podem ser alteradas a qualquer momento. Uma vez alteradas, todo o site é alterado automaticamente, ou seja, não há necessidade de ir página a página

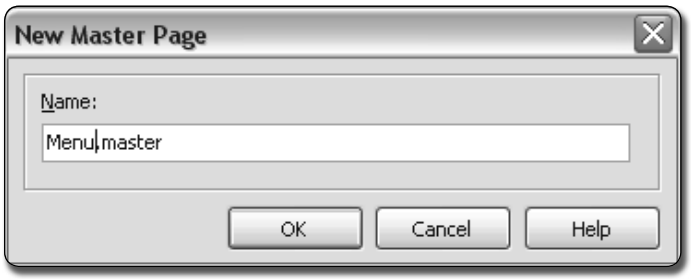


Figura 1. Criando uma Master Page chamada Menu

Figura 2. Formatando a Master Page

para fazer a sua atualização. Vamos ver na prática.

Crie um novo *Web Site* utilizando o menu *File>New>ASP.NET Web Application - Delphi for .Net* com o nome de *SiteMapExemplo*. Adicione um novo item usando o botão direito do mouse sobre o nome do projeto no *Project Manager* e em seguida em *Add New>Other>Delphi.NET Projects>New ASP.NET Files>Master Page*. Seu nome será *Menu.master* como mostra a **Figura 1**.

Uma *Master Page* nada mais é que uma página *ASP.NET*. A diferença é que uma *Master Page* vem com um controle chamado *ContentPlaceHolder* adicionado a ela. Iremos entender bem a dinâmica de uma *Master Page* assim que inserirmos novas páginas ao projeto. Por hora pense nessa página como sendo a página principal do site, e o controle *ContentPlaceHolder* é o espaço destinado ao conteúdo do site que será modificado enquanto navegamos.

Posicione o cursor antes do *ContentPlaceHolder* e adicione uma nova tabela à página, através do botão *Insert Table*. Insira uma tabela de uma linha e uma coluna. Arraste o *ContentPlaceHolder* para a célula central da tabela, como mostra a **Figura 2**. Realize as formatações de espaçamento, margens, cores etc., de forma que a página fique ao seu agrado. Leve em consideração que esta será a identidade visual de todo o seu site, e que o controle *ContentPlaceHolder* deve estar posicionado onde o conteúdo do site irá aparecer.

Depois de formatar a sua *Master Page*, crie um novo *Web Form* através da opção *Add New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Content Page*. Escolha o nome *Inicio.aspx* e observe que o RAD Studio já traz como default a *Master Page* que criamos (**Figura 3**).

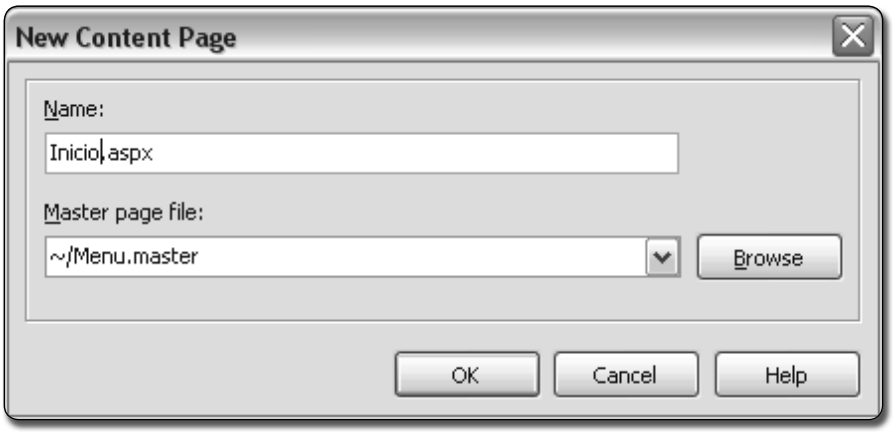


Figura 3. Adicionando uma página de conteúdo

Veja no *design* da página que a única área em que podemos trabalhar é justamente o espaço reservado pelo *ContentPlaceHolder* da *Master Page*. Nossa página de conteúdo fica revestida com as formatações realizadas na *Master Page*. Na página *Inicio.aspx*, inclua um texto para identificar a página. Se quiser pode realizar qualquer outro tipo de formatação ou incluir outros controles, o comportamento desta página é igual ao de uma página *.ASPX* tradicional.

Para prosseguirmos com o nosso exemplo, crie mais algumas páginas de conteúdo. Neste exemplo estamos criando as seguintes páginas: *Futebol.aspx, SaoPaulo.aspx, Palmeiras.aspx, Corinthians.aspx, XVdePiracicaba.aspx, Formula1.aspx, Ferrari.aspx, Willians.aspx* e *Tenis.aspx*. Inclua algum texto em cada uma destas páginas apenas para diferenciarmos uma das outras, se preferir pode adicionar controles, imagens etc. Por fim, na *Project Manager*, clique com o botão direito sobre a página *Inicio.aspx* e clique em *Set As Start Page*.

Site Map

O *ASP.NET 2.0* nos trouxe um novo objeto chamado *Site Map*, que é utilizado para cadastrarmos e organizarmos a estrutura hierárquica das páginas de um site. Para criar um *Site Map*, vá até a opção *Add Add New>Other>Delphi for .NET Projects>New ASP.NET Files>Site Map*. Um arquivo *XML* chamado *Web.sitemap*

Nota do DevMan

É possível utilizarmos mais de uma *Master Page* em um mesmo projeto. Por exemplo: digamos que nosso Web site possuia duas áreas distintas. Uma delas diz respeito a interface com o usuário final, ou seja, o internauta que acessa nosso site. A outra área é utilizada apenas para os desenvolvedores e administradores do site. Nessa área, os técnicos podem inserir dados, excluir informações, adicionar novos usuários, fazer backup do site e do banco de dados e uma série de outras rotinas comum aos administradores. Obviamente que essa área terá um menu diferenciado, logo é perfeitamente viável criar uma nova *Master Page* para que seja possível padronizar também essa área.

Listagem 1. Definindo a estrutura do Site

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Inicio.aspx" title="Inicio" description="Página Inicial" >
    <siteMapNode url="Futebol.aspx" title="Futebol">
      <siteMapNode url="SaoPaulo.aspx" title="São Paulo"/>
      <siteMapNode url="Palmeiras.aspx" title="Palmeiras"/>
      <siteMapNode url="Corinthians.aspx" title="Corinthians"/>
      <siteMapNode url="XVdePiracicaba.aspx" title="XV de Piracicaba"/>
    </siteMapNode>
    <siteMapNode url="Formula1.aspx" title="Fórmula 1">
      <siteMapNode url="Ferrari.aspx" title="Ferrari"/>
      <siteMapNode url="Willians.aspx" title="Willians"/>
    </siteMapNode>
    <siteMapNode url="Tenis.aspx" title="Tenis"  description="" />
  </siteMapNode>
</siteMap>
```

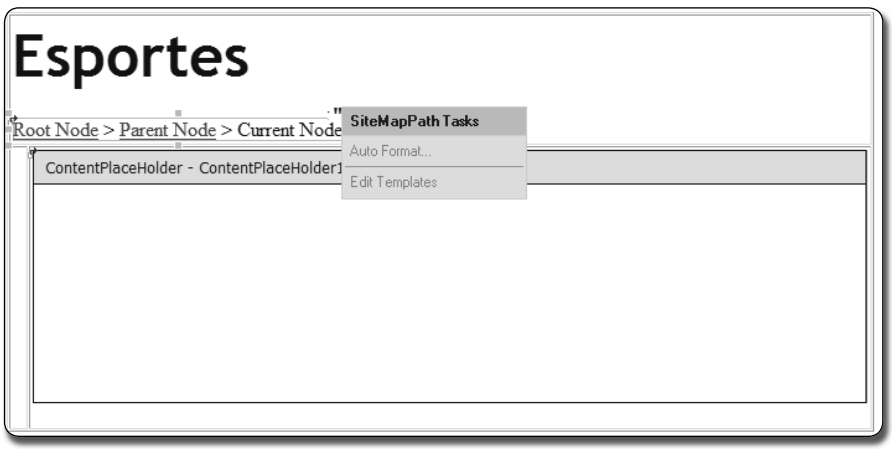


Figura 4. Incluindo o SiteMapPath na Master Page

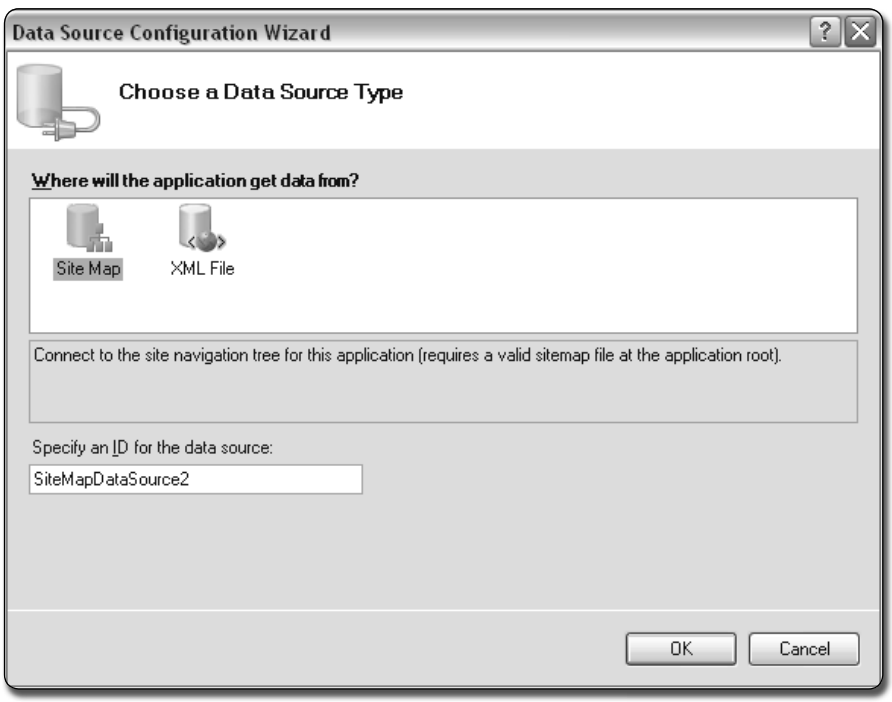


Figura 5. Definindo o Site Map como Data Source do menu

será criado. Observe que ele tem alguns elementos chamados *siteMapNode*. É utilizando este *XML* que iremos definir a estrutura do Site. Modifique o *Site Map* para que fique como demonstrado na **Listagem 1**.

Em um node *siteMapNode* temos que definir a *URL* da página, um título para a mesma e opcionalmente podemos definir uma descrição. Veja que colocamos os times de futebol dentro do nó *Futebol*, e as equipes de *Formula 1* dentro do nó *Formula1*. Dessa forma estamos criando a estrutura hierárquica do site. Podemos definir uma estrutura com quantos níveis desejarmos.

SiteMapPath

O *ASP.NET 2.0* também trouxe um conjunto de controles voltados para navegação. Expanda na barra de componente o item *Web Navigation* e veja os controles de navegação.

O *SiteMapPath* é um controle que mostra em uma estrutura hierárquica qual é a página atual. Vamos incluir um *SiteMapPath* em nossa *Master Page*. Para isso, inclua uma linha abaixo do título da tabela, e nessa linha arraste um *SiteMapPath* (**Figura 4)**.

Assim como a maioria de controles do *ASP.NET*, o *SiteMapPath* tem a opção *Auto Format* disponível em seu menu *Tasks*. Inclua a formatação da sua escolha no controle, aqui estamos utilizando o formato *Classic*.

Menu

Vamos agora incluir um controle de *Menu* na *Master Page*. Arraste um controle *Menu* na célula da esquerda da tabela e através das *Tasks*, altere o formato para *Classic*. Expanda a lista da opção *Choose Data Source* e clique em *new data source*. Aparecerá a janela de definição do *Data Source* (**Figura 5)**.

Nesta janela escolha a opção *Site Map*. O *Data Source* irá apontar automaticamente para o objeto *Site Map* que criamos *(web.sitemap)*, e que deve estar no *site root*. Em seguida clique em OK e execute o seu projeto. Verifique o resultado como mostra a **Figura 6**. Escolha as opções e veja que o conteúdo do site é modificado de acordo com as opções que escolhemos. Verifique também como o *SiteMapPath*

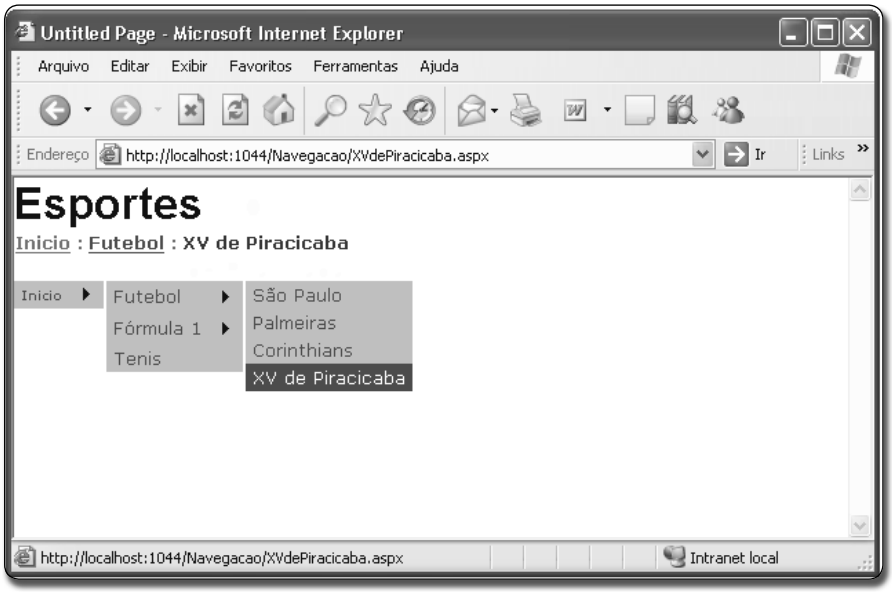


Figura 6. Resultado do controle Menu

Figura 7. Adicionando um controle TreeView à Master Page

é modificado automaticamente quando navegamos, mostrando o caminho completo da página atual.

TreeView

Por fim, vamos incluir uma *TreeView* logo abaixo do controle *Menu*. Altere o formato do controle *TreeView*, neste exemplo colocamos o formato *Arrows*. Na opção *Choose Data Source*, escolha o *SiteMapDataSource1*. Como já temos um *Data Source* configurado para o controle *Menu*, podemos aproveitá-lo para o *TreeView*. Veja que ao definir o *Data Source*, as opções do *TreeView* já são automaticamente preenchidas. Execute o projeto e verifique o resultado, como mostra a **Figura 7**.

Experimente navegar pelo site através da *TreeView*.Veja que ao escolhermos uma opção pela *TreeView*, automaticamente a respectiva opção selecionada do *Menu* é modificada. O *SiteMapPath* também é modificado independente de qual controle utilizamos para navegar.

Conclusão

Quantas linhas de código utilizamos? Absolutamente nenhuma. Com o uso de uma *Master Page*, três controles e um arquivo *Site Map* de configuração, criamos uma estrutura de navegação completa. É claro que em suas aplicações você irá optar por usar ou o *Menu* ou o *TreeView* para listar as opções. Neste exemplo incluímos os dois controles para que você possa verificar o comportamento de cada um deles. Quanto às opções de formatação que demonstramos, foram os *templates* que o *RAD Studio 2007* nos oferece. Através das propriedades dos controles podemos realizar alterações de layout mais avançadas. O importante é que podemos criar e customizar uma interface de navegação de maneira simples e rápida, sem a necessidade de codificações adicionais. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Sistema DataCar em OO

Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 4

Resumo DevMan

Como criar relatórios orientados a objetos? Veremos como fazer isso nesse artigo, dessa forma fecharemos com chave de ouro. Veremos como criar relatórios e como criar novas telas de acesso aos dados.

Nesse artigo veremos

- Criação de pesquisas usando OO;
- Criação de relatórios usando OO.

Qual a finalidade

- Neste artigo serão abordados os conceitos para criação e manutenção de código-fonte orientado a objetos.

Quais situações utilizam esses recursos?

- Em todo tipo de programa é perfeitamente possível adequar aos conceitos de programação OO.

Finalmente concluímos nossa caminhada e terminaremos nosso sistema. Fazendo uma breve retrospectiva, posso dizer que vimos uma ponta do que é o desenvolvimento orientado a objetos. Vimos que tudo começa por entender as regras do negócio e modelá-las, posteriormente codificá-las e então aplicar testes para que tudo fique assegurado. Depois então pensamos na persistência das informações e depois na boa estrutura que a aplicação deve ter, colocando tudo em sua respectiva camada. Nesta parte final também não será diferente. Vamos continuar com o desenvolvimento OO.

Paulo Roberto Quicoli
(pauloquicoli@gmail.com)
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP. Blog: http://pauloquicoli.spaces.live.com

Veículos procurados

Imagine que você está interessado em comprar um veículo, da *Volkswagem*, modelo *Golf*. Então você procura algumas revendas de automóveis e preenche um cadastro comprovando sua intenção. De posse desse cadastro, essa revenda, ao receber um veículo que se enquadra nas suas especificações fará um contato convidando-o a ver esse veículo. Esse é o cenário que vamos implementar, já possuímos uma classe *TVeiculoProcurado* e uma classe *TCliente*. Vamos imaginar aqui a interface com o usuário para

essa situação. Digamos que teremos um formulário inicial já exibindo todos os veículos procurados e que nesse mesmo formulário haja opções para excluir ou incluir um novo veículo procurado. Com isso em mente vamos criar um *Data Module* específico para essa situação.

Adicione um novo formulário ao projeto, salvando-o como *dmVeiculosProcurados*. Nele adicione um *TClientDataSet* e um *TDataSource*, chamando-os respectivamente de *cdsListaVeiculos* e *dsListaVeiculos*. Esse *ClientDataSet* será nosso *DTO* que exibirá a lista de veículos procurados que será retornada pelo seu respectivo *DAO*. Clique duas vezes no *cdsListaVeiculos* para que possamos acessar seus *Fields Editor*. Clique então com o botão direito e acesse *New Field*. Veja que estamos adicionando um novo campo ao *ClientDataSet*. Essa operação já é bem conhecida, portanto informe o nome do campo, tipo e tamanho conforme a **Tabela 1.**

Campo	Tipo	Tamanho
IdVeiculo	Integer	-
Modelo	String	100
Ano	Integer	-
Cor	String	50
Valor	Currency	-
Contato	String	150
IdCliente	Integer	

Tabela 1. Campos novos no cdsListaVeiculos

Agora é preciso incluir na classe *TVeiculoProcuradoDao* o método que irá nos retornar todos os veículos com essas informações. Abra a *unit Daodbx* e *DaoU*. Em *DaoU* localize a interface *IVeiculoProcuradoDao* e adicione uma função chamada *ListarTodosProcurados* que retorne um *TObjectList*. Na **Listagem 1** vemos a interface já alterada.

Agora é preciso implementar essa função na classe *TVeiculoProcuradoDao*. Na **Listagem 2** temos já o código implementado. Veja que não há segredos. Primeiro declaramos algumas variáveis que receberão a referência a determinados objetos que criaremos. Em seguida, montamos a instrução *SQL* que trará os dados que precisamos. Então ins-

Listagem 1. IVeiculoProcuradoDao

```
  IVeiculoProcuradoDao = interface(IDao)
    ['{744B08CF-528F-452B-89F3-EFD103BFA2E7}']
    procedure Excluir(const obj: TVeiculoProcurado);
    function Recuperar(id: Integer): TVeiculoProcurado;
    procedure Salvar(obj: TVeiculoProcurado);
    function ListarTodosProcurados: TObjectList;
  end;
```

Listagem 2. Método ListarTodosProcurados

```
function TVeiculoProcuradoDao.ListarTodosProcurados: TObjectList;
var
  veiculo: TVeiculoProcurado;
  modeloDao: TModeloDao;
  corDao: TCorDao;
  acessorioDao: TAcessorioDao;
  acessorio: TAcessorio;
  clienteDao: TClienteDao;
  ListaObjetos: TObjectList;
  sqlAcessorios: TSQLQuery;
begin
  SqlQuery.Close;
  SqlQuery.SQL.Clear;
  SqlQuery.SQL.Add('SELECT ID_VEICULO, ID_MODELO, ID_COR, ANOMODELO, ANOFABRICACAO,');
  SqlQuery.SQL.Add('COMBUSTIVEL, NUMERO_PORTAS, DATA_ENTRADA, VALOR_MAXIMO, ID_CLIENTE,
TIPO_VEICULO');
  SqlQuery.SQL.Add('FROM VEICULO WHERE TIPO_VEICULO = 0');
  try
    try
      SqlQuery.Open;
      if SqlQuery.IsEmpty then
         ListaObjetos := nil
      else
      begin
        ListaObjetos := TObjectList.Create;
        while not SqlQuery.Eof  do
        begin
          veiculo := TVeiculoProcurado.Create;
          veiculo.Id := SqlQuery.FieldByName('ID_VEICULO').asInteger;

          modeloDao := TModeloDao.Create;
          veiculo.Modelo := modeloDao.Recuperar(SqlQuery.FieldByName('ID_MODELO').AsInteger);

          corDao := TCorDao.Create;
          veiculo.Cor := corDao.Recuperar(SqlQuery.FieldByName('ID_COR').asInteger);
          clienteDao := TClienteDao.Create;

          veiculo.Cliente := clienteDao.Recuperar(SqlQuery.FieldByName('ID_CLIENTE').asInteger);

          veiculo.AnoModelo := SqlQuery.FieldByName('ANOMODELO').asInteger;
          veiculo.AnoFabricacao := SqlQuery.FieldByName('ANOFABRICACAO').AsInteger;
          veiculo.Combustivel := SqlQuery.FieldByName('COMBUSTIVEL').AsInteger;
          veiculo.NroPortas := SqlQuery.FieldByName('NUMERO_PORTAS').AsInteger;
          veiculo.DataEntrada := SqlQuery.FieldByName('DATA_ENTRADA').AsDateTime;
          veiculo.ValorMaximo := SqlQuery.FieldByName('VALOR_MAXIMO').AsCurrency;

          acessorioDao := TAcessorioDao.Create;

          SqlAcessorios := TSQLQuery.Create(nil);
          SqlAcessorios.SQLConnection := SqlQuery.SQLConnection;
          SqlAcessorios.Close;
          SqlAcessorios.SQL.Clear;
          SqlAcessorios.Params.Clear;
          SqlAcessorios.SQL.Add('SELECT ID_ACESSORIO FROM ACESSORIO_VEICULO WHERE ID_VEICULO = '
            + IntToStr(veiculo.Id));
          SqlAcessorios.Open;
          while not SqlAcessorios.Eof do
          begin
            acessorio := acessorioDao.Recuperar(SqlAcessorios.FieldByName('ID_ACESSORIO').AsInteger);
            veiculo.AddAcessorio(acessorio);
            SqlAcessorios.Next;
          end;
          SqlAcessorios.Close;
          FreeAndNil(SqlAcessorios);
          ListaObjetos.Add(veiculo);
          sqlQuery.Next;
        end;
      end;
    except
    on e:exception do
    begin
      raise Exception.Create('Falha ao recuperar veículos'+ #13+'Mensagem original: '+ e.Message);
      ListaObjetos := nil;
    end;
    end;
   finally
     SqlQuery.Close;
     result := ListaObjetos;
  end;
end;
```

Listagem 3. Classe TListaVeiculoProcuradoController

```
TListaVeiculoProcuradoController = class(TInterfacedObject, IController)
 private
    daoVeiculoProcurado: IVeiculoProcuradoDao;
 public
    constructor Create; reintroduce;
    procedure Initialize;
    procedure Finalize;
 end;
```

Listagem 4. TListVeiculosProcuradosViewF

```
  TListVeiculosProcuradosViewF = class(TListBaseF, IView)
  private
    { Private declarations }
    FController: TListaVeiculoProcuradoController;
  public
    { Public declarations }
    constructor Create(Owner: TComponent; Controller:
       TListaVeiculoProcuradoController); reintroduce;
    procedure Initialize;
  end;
```

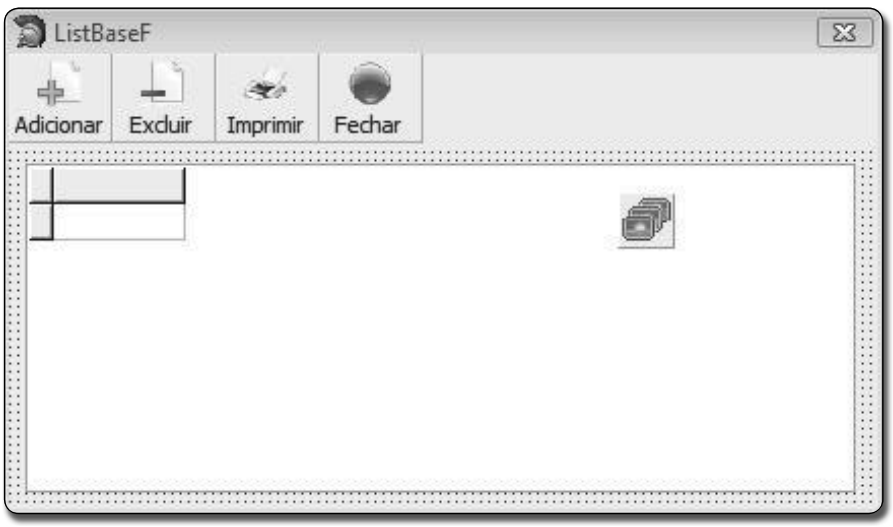


Figura 1. Formulário base para listar objetos

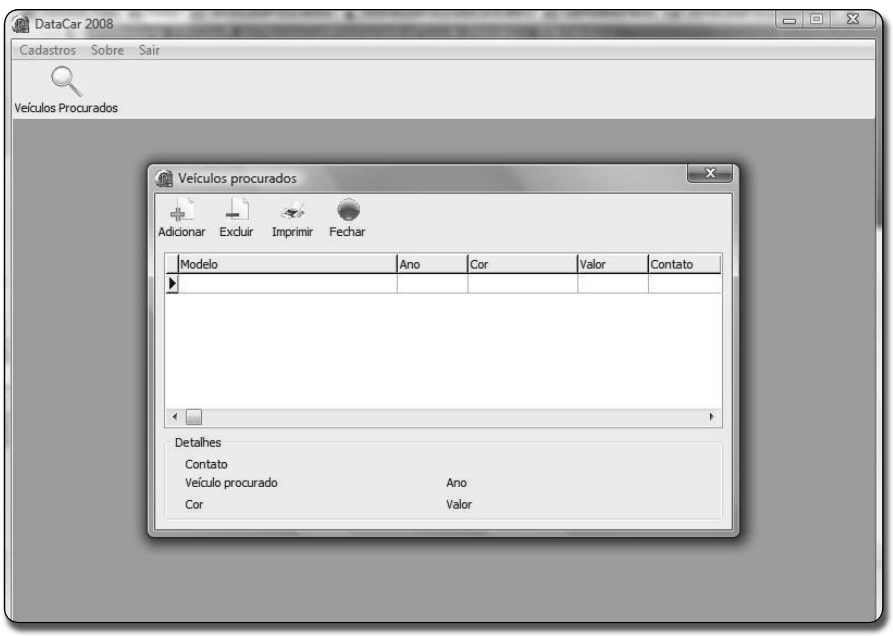


Figura 2. Veículos procurados

tanciamos a classe *TVeiculoProcurado* e mais a baixo atribuímos a ela os dados necessários para busca. Fazemos igual com a classe *TAcessorioDao*.

A interface para veículos procurados

Como fizemos no último artigo vamos começar a desenvolver nosso respectivo *Controller*. Adicione uma nova *Unit* ao projeto e salve-a como *listaVeiculoProcuradoControllerU.pas* e adicione as seguintes referências: *MVCInterfaces, DaoU, DaoDbx, ModelU, dmVeiculosProcuradosU, Contnrs*. Crie agora a classe que será nosso *controller*, como na **Listagem 3**.

O método *Create* simplesmente instancia o objeto *daoVeiculoProcurado* e inicia nosso objeto de transferência, o *dmVeiculosProcurados.cdsListaVeiculos* que posteriormente é fechado pelo método *Finalize*. O destaque fica no método *Initialize* que solicita ao *daoVeiculoProcurado* a lista desses veículos e preenche com essas informações o *cdsListaVeiculos*.

Esta *View* não herdará de *TcadBaseF* porque ela não é diretamente a tela de cadastros e sim uma tela de listagem de objetos, portanto vamos criar um novo formulário para que possamos reutilizá-lo na seqüência.

Adicione um novo formulário ao projeto, salvando sua *Unit* como *ListBaseU.pas*, seu formulário como *ListBaseF* e o remova da lista *Auto Create forms*. Na **Figura 1** vemos o formulário pronto.

Vamos agora criar a *View* que listará os veículos procurados. Adicione um novo formulário que herde de *TListBaseF* e

Nota do DevMan

Muitos iniciantes não percebem a finalidade de um *datamodule* e acabam utilizando apenas um para toda sua aplicação, dificultando a localização de controles e consequentemente sua manutenção. Um *datamodule* existe para ser utilizado como uma ferramenta modular de dados, seu nome realmente quer dizer "modulo de dados". Dessa forma podemos criar quantos *datamodules* forem necessários para organizar nossos dados. Uma sugestão é organizá-los por funções específicas dentro do sistema, como se estivesse criando "setores".

salve-o como *ListVeiculosProcuradosView* e já o remova da lista *Auto Create forms*.

A esse novo formulário adicione referências à *MVCInterfaces, dmVeiculosProcuradosU* e *listaVeiculoProcuradoControllerU*. Na **Listagem 4** vemos a estrutura da nossa *View*.

Como fizemos em outras *Views* o método *Initialize* executa o *Initialize* de seu respectivo *Controller*.

Executando a View

Adicionar um novo veículo procurado é algo que pode ser rotineiro no dia-a-dia de uma revenda de carros por isso vamos adicionar um atalho para essa função no formulário principal. Adicione um botão à *ToolBar* e faça referência às *units listaVeiculoProcuradoControllerU* e *ListVeiculosProcuradosViewU*. No *OnClick* desse botão vamos executar nossa *View* como é mostrado na **Listagem 5.** E na **Figura 2** vemos o formulário principal e nossa *view* em execução.

Adicionando um novo veículo

Para adicionar um novo veículo procurado vamos primeiramente criar o *DTO* que nos auxiliará. Adicione um novo *TClientDataSet* ao *Data Module*, crie os campos conforme **Figura 3** e também ligue a ele um novo *TDataSource*.

Nosso próximo passo então é criar o *Controller* responsável pela inclusão de um novo veículo procurado. Adicione uma nova *unit* ao projeto, salvando-a como *cadVeiculoProcuradoControllerU.pas*. Crie uma nova classe que herde de *IController*. Na **Listagem 6** temos a definição do *Controller*.

Semelhantemente ao que fizemos nos *Controllers* anteriores no método *Initialize* vamos carregar os objetos *DTO*, que neste caso são *dmCadastros.cdsModelo* e *dmCadastros.cdsCor* reutilizando assim os *ClientDatasets* existentes.

Os métodos *Insert*, *Edit* e *Cancel* realizam essas operações no *cdsVeiculoProcurado* enquanto que o *Finalize* fecha os *ClientDataSets* abertos anteriormente. Quanto ao método *Save*, ele é encarregado de instanciar um objeto do tipo *TVeiculoProcurado* e então a partir dos *DTOs* obter os valores necessários e salvar o objeto através de seu *DAO*, como é visto na **Listagem 7**.

Listagem 5. Executando a View de veículos procurados

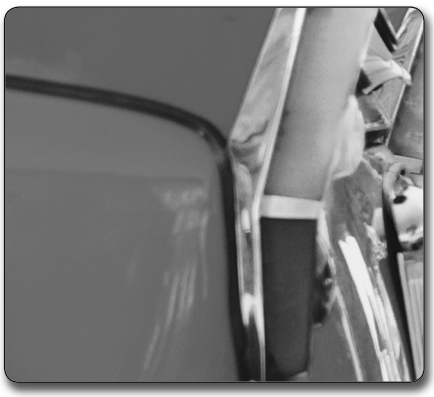
```
procedure TPrincF.btVeiculosProcuradosClick(Sender: TObject);
var
  Controller: TListaVeiculoProcuradoController;
begin
  Controller := TListaVeiculoProcuradoController.Create;
  try
    ListVeiculosProcuradosViewF := TListVeiculosProcuradosViewF.Create(nil, Controller);
    ListVeiculosProcuradosViewF.ShowModal;
  finally
    FreeAndNil(ListVeiculosProcuradosViewF);
  end;
end;
```

Listagem 6. TcadVeiculoProcuradoController

```
type
  TCadVeiculoProcuradoController = class(TInterfacedObject, IController)
    private
      daoVeiculoProcurado: IVeiculoProcuradoDao;
      daoCor: ICorDao;
      daoCliente: IClienteDao;
      daoModelo: IModeloDao;
    public
      constructor Create; reintroduce;
      procedure Initialize;
      procedure Insert;
      procedure Edit;
      procedure Save;
      procedure Finalize;
      procedure Cancel;
  end;
```

Listagem 7. Salvando um veículo procurado

```
procedure TCadVeiculoProcuradoController.Save;
var
  objModelo: TModelo;
  objCor: TCor;
  objCliente: TCliente;
  objVeiculo: TVeiculoProcurado;
begin
  objModelo := TModelo.Create;
  objModelo.Id := dmCadastros.cdsModeloIdModelo.AsInteger;

  objCor := TCor.Create;
  objCor.Id := dmCadastros.cdsCorIdCor.AsInteger;

  objCliente := TCliente.Create;
  objCliente.Id := dmVeiculosProcurados.cdsClienteIdCliente.AsInteger;

  objVeiculo := TVeiculoProcurado.Create;
  objVeiculo.Cliente := objCliente;
  objVeiculo.Modelo := objModelo;
  objVeiculo.Cor := objCor;

  objVeiculo.DataEntrada := now;
  objVeiculo.ValorMaximo := dmVeiculosProcurados.cdsVeiculoProcuradoValor.AsCurrency;
  objVeiculo.AnoFabricacao := dmVeiculosProcurados.cdsVeiculoProcuradoAno.AsInteger;
  objVeiculo.AnoModelo := dmVeiculosProcurados.cdsVeiculoProcuradoAno.AsInteger;
  objVeiculo.NroPortas := dmVeiculosProcurados.cdsVeiculoProcuradoPortas.AsInteger;
  objVeiculo.Combustivel := COMB_FLEX;

  try
    try
      daoVeiculoProcurado.Salvar(objVeiculo);
  dmVeiculosProcurados.cdsVeiculoProcuradoIdVeiculoProcurado.AsInteger := objVeiculo.Id;
      dmVeiculosProcurados.cdsVeiculoProcurado.Post;
    except
    on e: exception do
      begin
        dmVeiculosProcurados.cdsVeiculoProcurado.Cancel;
        raise Exception.Create(e.Message);
      end;
    end;
  finally
    FreeAndNil(objVeiculo);
  end;
end;
```



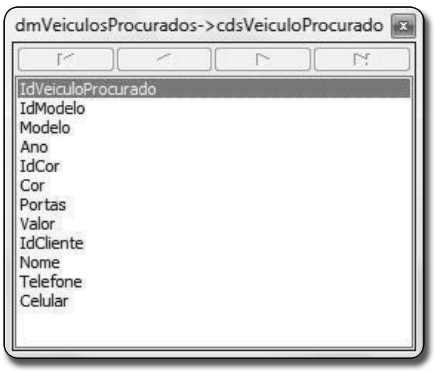


Figura 3. DTO cdsVeiculoProcurado

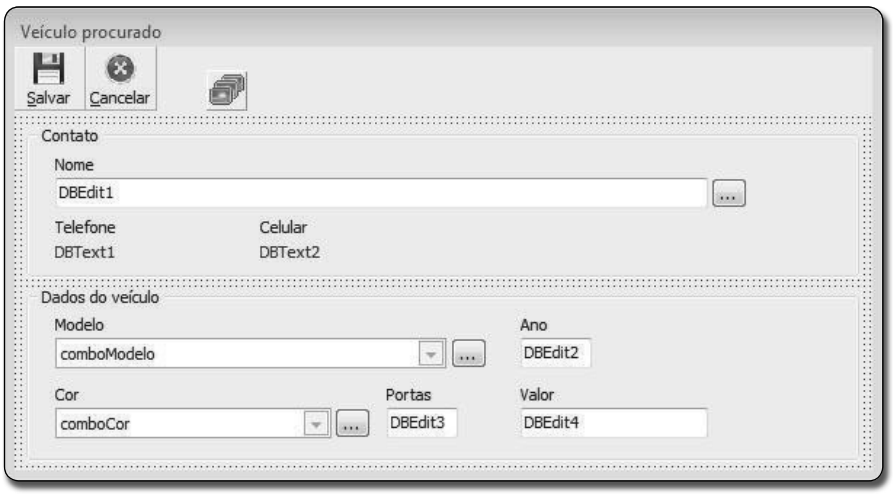


Figura 4. Cadastro de veículos procurados

Listagem 8. Permitindo o cadastro de um novo modelo

```
procedure TEditVeiculoProcuradoViewF.btLocalizaModeloClick(Sender: TObject);
var
 ControllerModelo: TCadModeloController;
begin
  inherited;
  ControllerModelo := TCadModeloController.Create;
  ControllerModelo.Edit;
  EditModeloF := TEditModeloF.Create(nil,ControllerModelo);
  try
    EditModeloF.ShowModal;
  finally
    FreeAndNil(EditModeloF);
    FController.Initialize;
  end;
end;
```

Listagem 9. Ajustanto o método Create

```
constructor TCadModeloController.Create;
begin
  daoMarca := TMarcaDaoDbx.Create;
  try
    dmCadastros.cdsMarca.Close;
    dmCadastros.cdsMarca.CreateDataSet;
  except
    dmCadastros.cdsMarca.EmptyDataSet;
    dmCadastros.cdsMarca.Open;
  end;

  daoModelo := TModeloDao.Create;
  try
    dmCadastros.cdsModelo.Close;
    dmCadastros.cdsModelo.CreateDataSet;
  except
    dmCadastros.cdsModelo.EmptyDataSet;
    dmCadastros.cdsModelo.Open;
  end;
end;
```

View de Cadastro de Veículos Procurados

Adicione um novo formulário ao projeto, herdando de *TEditBase*. Salve-o como *EditVeiculoProcuradoView* e o configure como na **Figura 4**.

Ligue os controles *DBWare* aos campos do *cdsVeiculoProcurado*, sendo que para os controles *comboModelo* e *comboCor* ajuste apenas as propriedades de exibição da lista. Observe que ao lado de cada *dbLookupComboBox* existe um botão que será utilizado para permitir a inclusão de um novo modelo ou nova cor caso seja necessário. No clique desse botão nós chamamos o *controller* e a respectiva *view* de edição (ver **Listagem 8)**, porém para que tudo funcione perfeitamente é preciso realizar um pequeno ajuste no *Controller* de edição, conforme **Listagem 9**.

Localizando um cliente

É fundamental oferecer ao usuário a possibilidade de se procurar por clientes existentes ou de se criar um novo durante o cadastro de veículos procurados. Pensando nisso, na **Figura 5,** temos um formulário de pesquisa que também permite a inclusão de um novo cliente e sua devida transferência ao cadastro de veículos procurados.

Ao digitar o nome de um cliente e pressionar *ENTER*, o respectivo *controller* é então acionado para recuperar os clientes que possuem um nome iniciando com o texto informado. Na **Listagem 10** podemos ver que o *controller* preenche um *DTO* com o resultado do método *daoCliente.ListarPorNome*. Caso não exista

Nota

O cadastro de veículos procurados exige que um determinado veículo esteja relacionado a um cliente, dessa maneira para que possamos permitir a escolha do cliente temos que criar o cadastro de clientes. Semelhantemente ao que estamos fazendo, o cadastro de clientes também terá um formulário de lista que exibirá os clientes, e a partir dele se poderá realizar as operações de inclusão, edição ou exclusão. Devido a limites de espaço, não colocarei aqui todos os passos para a criação dessa rotina, mostrarei contudo como ela é utilizada no cadastro de veículos procurados. Vale lembrar que se o leitor sentir alguma dificuldade, os fontes deste projeto estão completos e disponíveis para download.

o cliente, então seu cadastro é permitido. Na **Figura 6** vemos a *view* de cadastro de clientes.

Com o cliente cadastrado e devidamente localizado é preciso relacioná-lo ao veículo procurado, para isso criamos no *controller* de *TListaClienteController* o método *Transferir*. Como podemos ver na **Listagem 11** o método *Transferir* apenas transfere informações entre nossos *DTO's*.

Imprimindo a lista de veículos procurados

Infelizmente não existem geradores de relatório que nativamente estejam preparados para trabalhar com objetos diretamente. No geral estão preparados para utilizar um *TDataset* como fonte de dados. Por isso que utilizamos como *DTO* um *TClientDataset*.

Para imprimir um relatório basta se criar, ou reutilizar, um *ClientDataSet* e preenchê-lo com as informações que desejamos imprimir. Aqui vou implementar nosso relatório utilizando um o *FastReport*, porém sinta-se à vontade para utilizar o de sua preferência.

Adicione um novo *Data Module* ao projeto, salvando-o como *dmRelatorios* e nele adicione um novo *TClientDataSet*, chamando-o de *cdsRelVeiculosProcurados* e adicione os seguinte campos, conforme visto na **Figura 7**. Crie um relatório semelhante ao da **Figura 8**.

MVC para relatórios?

Vamos aplicar também o *MVC* para criação de nossos relatórios. Vamos separar a interface de tal forma que ela não saiba se o relatório é do *FastReports*, *QuickReport*, *Rave* ou qualquer outro.

Adicione uma nova *unit* ao projeto, *RelatoriosInterfaces.pas*. Nela criamos uma interface básica a qualquer relatório que vamos implementar, não importando do que trata o relatório ou o gerador utilizado. Veja **Listagem 12.**

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula que mostra como utilizar o FastReport.

*www.devmedia.com.br/articles/viewcomp.asp?comp=3645&hl=*fastreport**

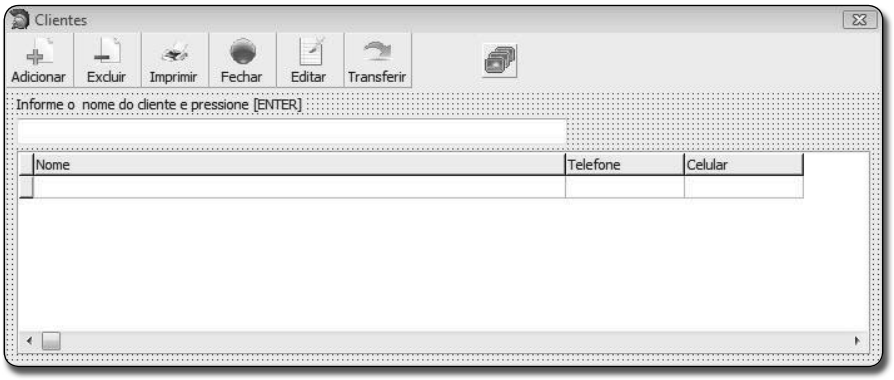


Figura 5. Pesquisando um cliente

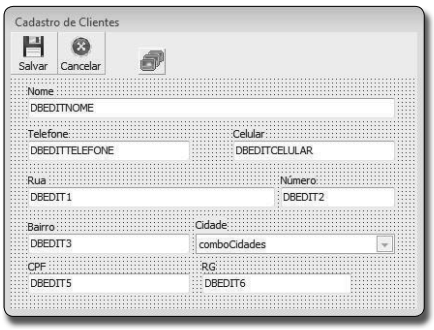


Figura 6. Cadastro de clientes

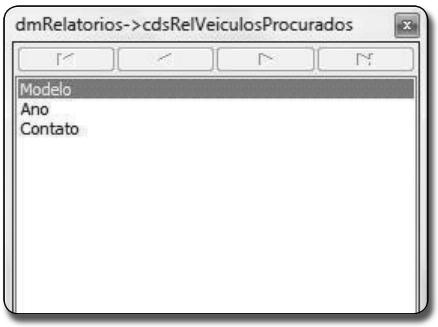


Figura 7. cdsRelVeiculosProcurados

Listagem 10. Localizando um cliente por seu nome

```
procedure TListaClienteController.LocalizarPorNome(Nome: string);
var
 I: integer;
 Cliente: TObjectList;
begin
  Cliente := daoCliente.ListarPorNome(Nome);
  try
    dmVeiculosProcurados.cdsCliente.Close;
    dmVeiculosProcurados.cdsCliente.CreateDataSet;
  except
    dmveiculosProcurados.cdsCliente.EmptyDataSet;
    dmVeiculosProcurados.cdsCliente.Open;
  end;

  if Cliente <> nil then
     for I := 0 to Cliente.Count - 1 do
       begin
         dmVeiculosProcurados.cdsCliente.Append;
         dmVeiculosProcurados.cdsClienteIdCliente.AsInteger :=  TCliente(Cliente[I]).Id;
         dmVeiculosProcurados.cdsClienteNome.AsString := TCliente(Cliente[I]).Nome;
         dmVeiculosProcurados.cdsClienteTelefone.AsString := TCliente(Cliente[I]).Telefone;
         dmVeiculosProcurados.cdsClienteCelular.AsString := TCliente(Cliente[I]).Celular;
         dmVeiculosProcurados.cdsCliente.Post;
       end;

end;
```

Listagem 11. Transferindo um cliente para um veículo

```
procedure TListaClienteController.Tranferir;
begin
  dmVeiculosProcurados.cdsVeiculoProcuradoIdCliente.AsInteger :=
    dmVeiculosProcurados.cdsClienteIdCliente.AsInteger;
  dmVeiculosProcurados.cdsVeiculoProcuradoNome.AsString :=
    dmVeiculosProcurados.cdsClienteNome.AsString;
  dmVeiculosProcurados.cdsVeiculoProcuradoTelefone.AsString :=
    dmVeiculosProcurados.cdsClienteTelefone.AsString;
  dmVeiculosProcurados.cdsVeiculoProcuradoCelular.AsString :=
    dmVeiculosProcurados.cdsClienteCelular.AsString;
end;
```

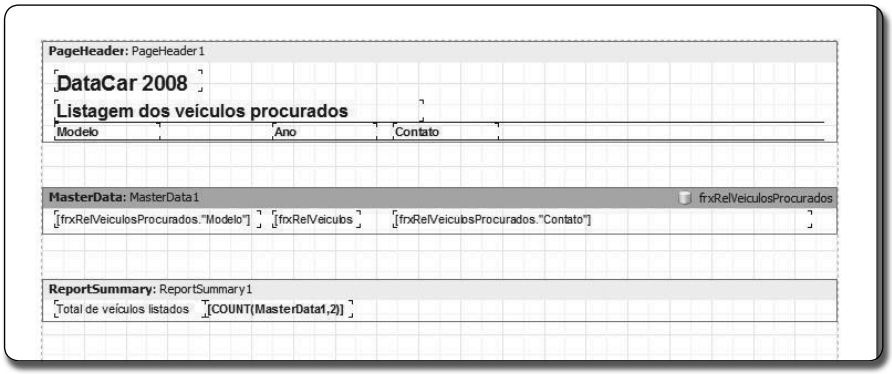


Figura 8. Listagem de veículos procurados

Listagem 12. Interface comum para os relatórios

```
unit RelatoriosInterfaces;

interface

type
  IRelatorioBase = interface
    ['{3EDB0EC2-4531-47E0-A36A-81DFDD6BBC1C}']
    procedure Preparar;
    procedure Imprimir;
    procedure Visualizar;
  end;

implementation

end.
```

Listagem 13. Classe de implementação

```
  TRelVeiculosProcurados = class(TInterfacedObject,IRelatorioBase)
    procedure Preparar;
    procedure Imprimir;
    procedure Visualizar;
  end;
```

Listagem 14. TrelVeiculosProcuradosController

```
  TRelVeiculosProcuradosController = class(TInterfacedObject, IController)
  private
    FGerador: IRelatorioBase;
    daoVeiculoProcurado: IVeiculoProcuradoDao;
  public
    constructor Create(Gerador: IRelatorioBase); reintroduce;
    procedure Initialize;
    procedure Imprimir;
    procedure Visualizar;
    procedure Finalizar;
  end;
```

Listagem 15. Carregando as informações para o relatório

```
procedure TRelVeiculosProcuradosController.Initialize;
var
 I: integer;
 Veiculos: TObjectList;
begin
  Veiculos := daoVeiculoProcurado.ListarTodosProcurados;
  if Veiculos <> nil then
     for I := 0 to Veiculos.Count - 1 do
       begin
         dmRelatorios.cdsRelVeiculosProcurados.Append;
         dmRelatorios.cdsRelVeiculosProcuradosModelo.AsString :=
           TVeiculoProcurado(Veiculos[I]).Modelo.Nome;
         dmRelatorios.cdsRelVeiculosProcuradosAno.AsString :=
           IntToStr(TVeiculoProcurado(Veiculos[I]).AnoModelo);
         dmRelatorios.cdsRelVeiculosProcuradosContato.AsString :=
           TVeiculoProcurado(Veiculos[I]).Cliente.Nome + ' Fone: ' +
           TVeiculoProcurado(Veiculos[I]).Cliente.Telefone + ' Celular: ' +
           TVeiculoProcurado(Veiculos[I]).Cliente.Celular;
         dmRelatorios.cdsRelVeiculosProcurados.Post;
       end;
  FGerador.Preparar;
end;
```

Adicione uma nova *Unit*, salvando-a como *RelatoriosFast.pas*. Nela adicione referencias à *dmRelatoriosU* e *RelatoriosInterface* e crie a classe contida na **Listagem 13**.

> **Nota:** Esta *unit* irá conter os relatórios implementados com o *FastReport*, de forma semelhante você pode criar uma *unit RelatoriosQuick* ou *RelatoriosRave* e nelas implementar a interface *IRelatorioBase* de acordo com o seu gerador.

O Controller

Nosso *controller* irá preencher nosso *DTO*, o *dmRelatorios.cdsRelVeiculosProcurados*, com as informações necessárias e como os relatórios apenas conhecem nosso *DTO*, eles estarão prontos para ler esse *ClientDataSet*. Adicione uma nova *unit* ao projeto, salvado-a como *RelVeiculosProcuradosController* e adicione referências às *units MVCInterfaces, RelatoriosInterfaces, daoDBX, DaoU, ModelU, dmRelatoriosU, Contnrs* e *SysUtils*. Então crie a classe *TRelVeiculosProcuradosController* como na **Listagem 14**.

O método mais importante dessa classe é o *Initialize*, pois é nele que os dados são obtidos. Veja como na **Listagem 15**. Nele nós instanciamos a classe *daoVeiculoProcurado* e atribuímos a ela todos veículos procurados, em seguida adicionamos ao *cdsRelVeiculosProcurados* os registros que faremos a impressão.

E a View?

Nosso relatório será acionado a partir de uma *view* já existente, a *ListClientesView*. Para que isso aconteça vamos adicionar referências à *RelVeiculosProcuradosController* e *RelatoriosFast* e posteriormente no *OnClick* do botão *imprimir* vamos instanciar o *controller*, como na **Listagem 16**, e na **Figura 9** temos o relatório finalizado.

Veja, passamos para o método *Create* uma instância de *TRelVeiculosProcurados*, que implementa o relatório utilizando o *FastReport*. Todas as características do *FastReport* estão encapsuladas nessa classe e são acessadas pelos métodos do *controller*, que é a única coisa que a *View*

conhece. Dessa forma podemos mudar o gerador de relatórios com o mínimo de alterações na *View*.

> **Nota:** Uma outra abordagem poderia ser aplicada quanto ao uso da *View*. Ao invés de utilizar a *View TListVeiculosProcuradosViewF*, poderíamos ter utilizado o seu *Controller* para chamada do *Controller* do relatório.

Listagem 16. Imprimindo a listagem

```
procedure TListVeiculosProcuradosViewF.tbImprimirClick(
  Sender: TObject);
var
 controller: TRelVeiculosProcuradosController;
begin
  inherited;
  controller := TRelVeiculosProcuradosController.Create(
    TRelVeiculosProcurados.Create);
  controller.Initialize;
  controller.Visualizar;
  controller.Finalizar;
  FreeAndNil(controller);
end;
```

Veículos para venda

Seguindo o padrão exibido para construção do item *Veículos procurados* vamos também criar o item *veículos para venda*, dessa forma teremos um formulário que irá listar os veículos para venda, permitindo uma busca nesses veículos, e que também irá permitir a inclusão de um novo veículo. Essa nova inclusão de veículo por sua vez deve permitir a inclusão de um proprietário.

A primeira coisa a se fazer é definir então na interface *IVeiculoParaVendaDao* um método chamado *ListaTodosParaVenda* e implementá-lo em *TVeiculoParaVendaDao*. Nesse método todos os veículos disponíveis para venda devem ser listados.

Então vamos criar nosso *DTO* que irá representar essa coleção de veículos para venda. Vamos criar um novo *Data Module* para isso, e chamá-lo de *dmVeiculosParaVenda*. Nele vamos adicionar um *TClientDataset* e ajustar sua propriedade *Name* para *cdsListaVeiculosName*, não esquecendo de adicionar também seu *TDatasource* correspondente. Então

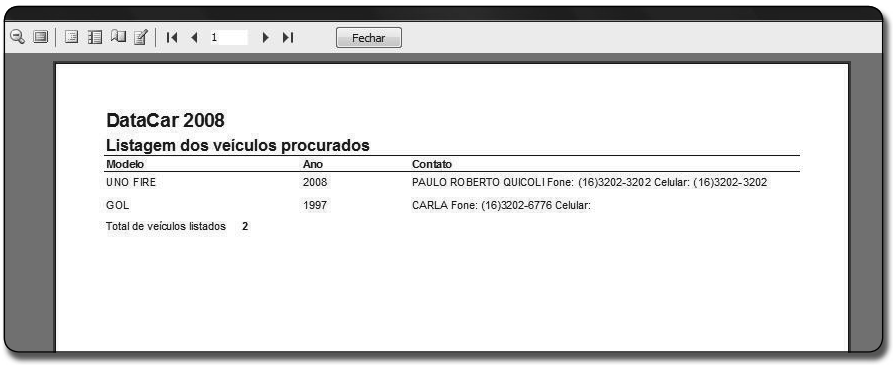


Figura 9. Relatório em execução

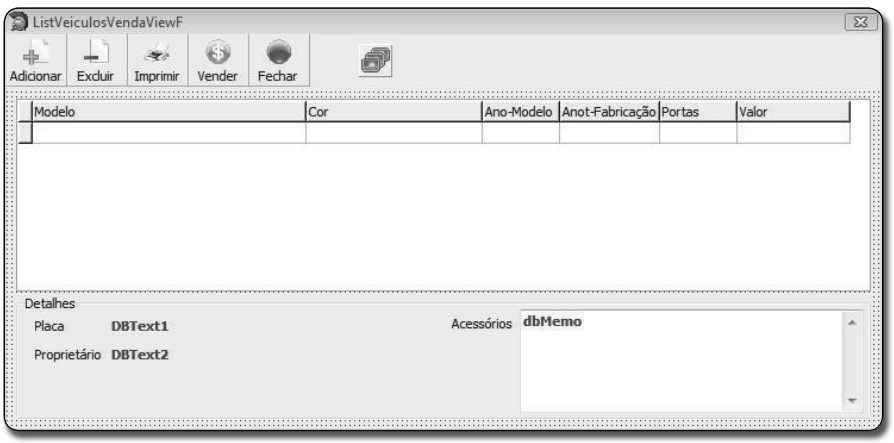


Figura 10. View de venda de veículos

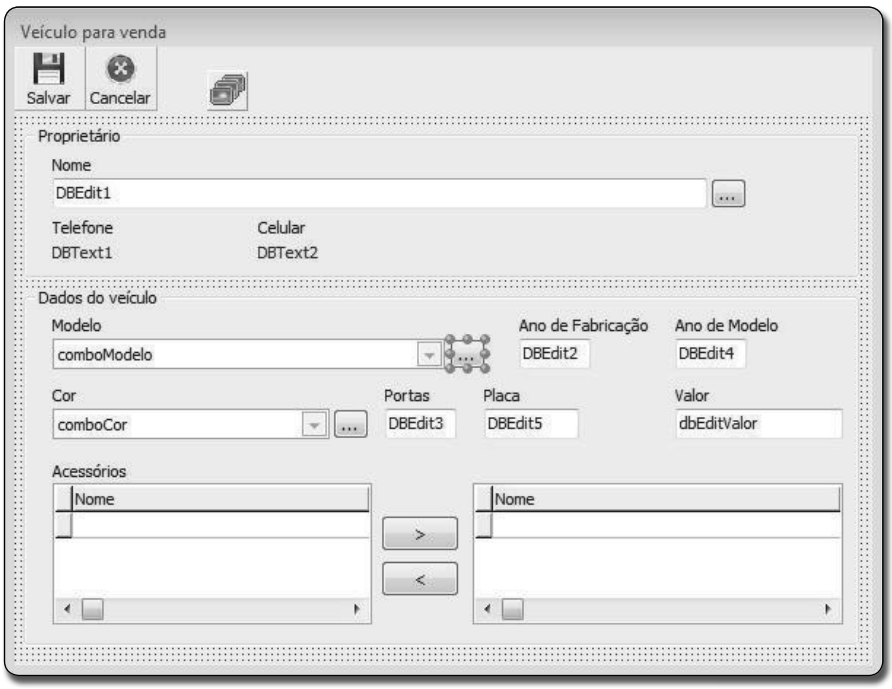


Figura 11. Cadastro de veículo para venda

Campo	Tipo	Tamanho
IdVeiculo	Integer	-
Modelo	String	100
AnoModelo	Integer	-
Cor	String	50
Valor	Currency	-
Proprietario	String	150
IdProprietario	Integer	-
AnoFabricacao	Integer	-
Placa	String	10
Portas	Integer	-
Acessorios	Memo	-

Tabela 2. Campos de cdsListaVeiculosName

Campo	Tipo	Tamanho
IdVenda	Integer	-
IdVeiculo	Integer	-
IdModelo	Integer	-
Modelo	String	50
Placa	String	10
Proprietario	String	150
ValorEsperado	Currency	-
ValorSugerido	Currency	-
ValorReal	Currency	10
IdComprador	Integer	-
Comprador	String	70

Tabela 3. Campos do cdsVeiculosParaVenda

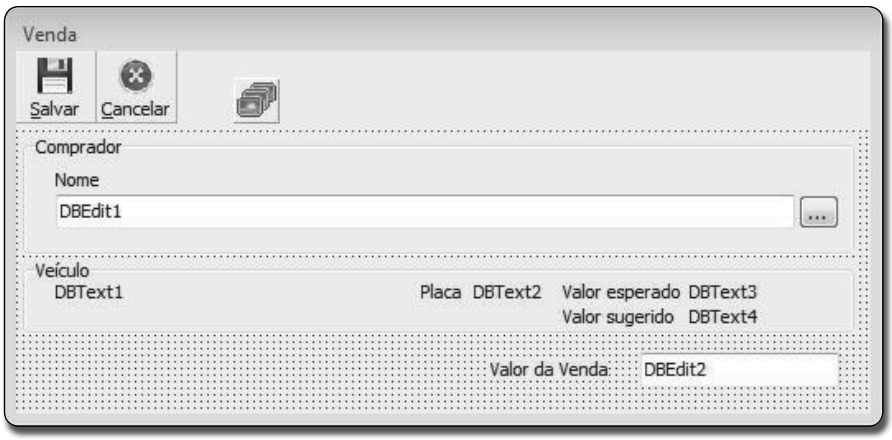


Figura 12. Venda de veículo

criamos os campos referentes a *veículos para venda*, conforme abaixo a **Tabela 2**. Repita os mesmos passos que fizemos para adicionar os campos ao *Field Editor* do *cdsVeiculosProcurados*. Veja nas **Figuras 10** e **11** a *View ListVeiculosVendaViewF* e *EditVeiculoVendaF*.

Realizando a Venda

Como vocês perceberam, na *view* que lista os veículos à venda temos um botão *Vender*. Nele vamos acionar a rotina que registra a venda do veículo selecionado. Vamos adicionar um *DTO* que irá nos auxiliar no processo de venda. Dentro do *dmVeiculosParaVenda* vamos adicionar um novo *TClientDataset(cdsVeiculosParaVenda)* e seu *TDataSource*. No *cdsVeiculosParaVenda* vamos adicionar campos que estão descritos na **Tabela 3**.

Controller

Após adicionar uma nova *unit* ao projeto e setar as referências necessárias criamos então nosso *controller* como na **Listagem 17.**

O método *Initialize* desse *controller* já insere um registro em nosso *DTO* com os valores do veículo selecionado, veja **Listagem 18.** Novamente, não há segredos, apenas instanciamos a classe e a usamos. Nesse caso, estamos adicionando ao *cdsVenda* os dados referentes a uma nova venda.

View

Nossa *View* de *Vendas* deve ser como a da **Figura 12**, que é então chamada pela *view* que lista os carros disponíveis para venda, conforme **Listagem 19.** É interessante, todas as nossas telas são bastante simples, porém totalmente funcionais. Na **Listagem 19**, estamos instanciando a classe e logo em seguida chamando o método *ShowModal* do formulário.

Conclusão

Nesta última parte do mini-curso concluímos o desenvolvimento do sistema *DataCar*, porém algumas funcionalidades não foram citadas para o escopo do sistema como por exemplo uma listagem do que foi vendido no mês, impressão da nota fiscal de venda, uma consulta mais detalhada dos veículos disponíveis

Listagem 17. Controller de Venda

```
  TCadVendaController = class(TInterfacedObject, IController)

    private
      daoVenda: TVendaDao;

    public
      constructor create; reintroduce;
      procedure Initialize;
      procedure Salvar;
      procedure Cancelar;
  end;
```

Listagem 18. Iniciando o controller

```
procedure TCadVendaController.Initialize;

var
 daoVeiculo: TVeiculoParaVendaDao;
 objVeiculo: TVeiculoParaVenda;

begin
  daoVeiculo := TVeiculoParaVendaDao.Create;
  objVeiculo := daoVeiculo.Recuperar(dmVeiculosParaVenda.
    cdsListaVeiculosVendaIdVeiculo.AsInteger);

  dmVeiculosParaVenda.cdsVenda.Insert;
  dmVeiculosParaVenda.cdsVendaIdVeiculo.AsInteger := objVeiculo.Id;
  dmVeiculosParaVenda.cdsVendaModelo.AsString := objVeiculo.Modelo.Nome;
  dmVeiculosParaVenda.cdsVendaIdModelo.AsInteger := objVeiculo.Modelo.Id;
  dmVeiculosParaVenda.cdsVendaPlaca.AsString := objVeiculo.Placa;
  dmVeiculosParaVenda.cdsVendaValorEsperado.
    AsCurrency := objVeiculo.ValorEsperadoVenda;
  dmVeiculosParaVenda.cdsVendaValorSugerido.
    AsCurrency := objVeiculo.ValorSugeridoVenda;
  dmVeiculosParaVenda.cdsVendaValorReal.AsCurrency :=
    objVeiculo.ValorRealVenda;
end;
```

Listagem 19. Chamando a venda de veículo

```
procedure TListVeiculosVendaViewF.btVenderClick(Sender: TObject);

var
 Controller: TCadVendaController;

begin
  Controller := TCadVendaController.create;
  CadVendaF := TCadVendaF.Create(nil,Controller);
  CadVendaF.ShowModal;
  FreeAndNil(CadVendaF);
  FController.Initialize;
end;
```

e até mesmo uma rotina que automaticamente notifica um cliente, através de e-mail, quando um veículo com as características que ele procura se torna disponível para venda. Essas rotinas complementares vocês mesmos podem desenvolver e qualquer dúvida, estou à disposição.

Sobre a metodologia utilizada, OO, vimos a mudança de pensamento que ela nos traz, além é claro de um pouco mais de trabalho. Apesar de termos empregado aqui o uso do padrão *DAO*, eu recomendo o uso de algum *framework* de persistência, para que dessa maneira possamos nos focar realmente nas regras de negócio.

Espero que não só com este artigo, mas com tantos outros já mostrados pela equipe da ClubeDelphi, o mito de se desenvolver OO vá diminuindo entre a comunidade e que juntos possamos evoluir e tirar o máximo de proveito que essa tecnologia nos oferece para assim prosseguir para assuntos mais avançados como *IoC* (*Inversion of Control*) e *AOP*(*Aspect Oriented Programming*). Muito obrigado. ●

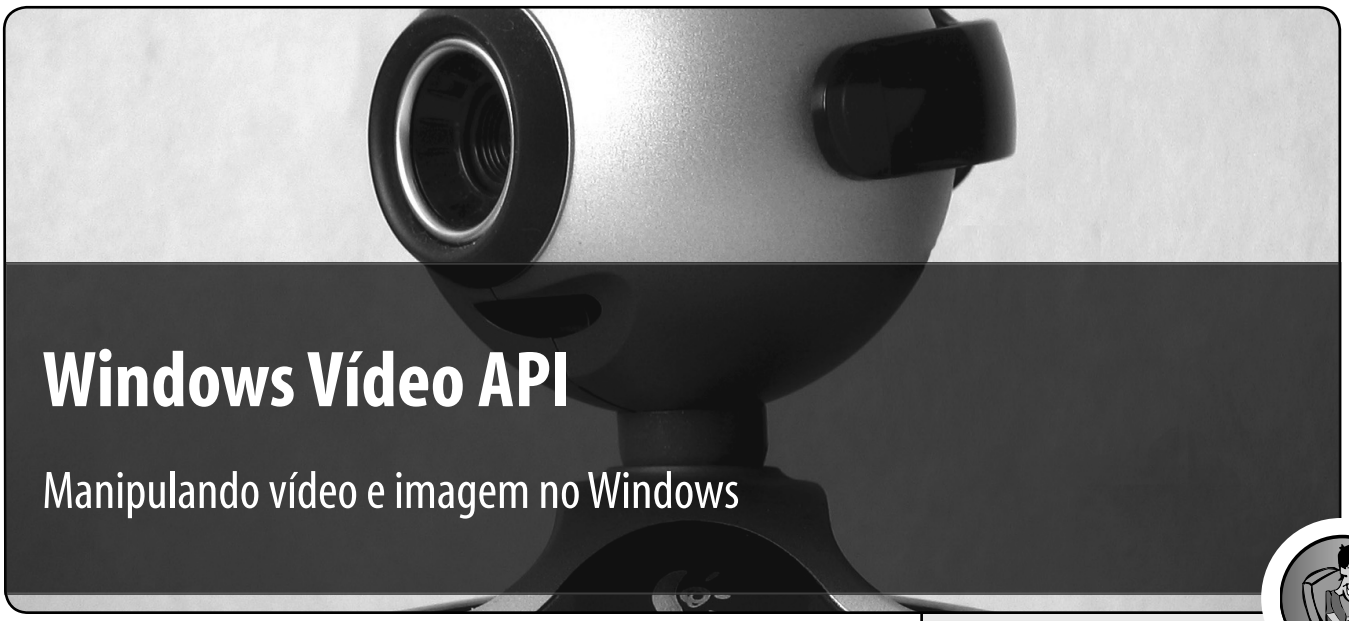
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET



Windows Vídeo API

Manipulando vídeo e imagem no Windows

Resumo DevMan

Esse artigo visa mostrar os principais recursos da API do Windows para captura e manipulação de imagens utilizando Webcams.

Nesse artigo veremos

- API de Vídeo do Windows;
- Como funciona o método capCreate CaptureWindowA da AVICAP32.DLL;
- Como enviar mensagens para uma webcam ou câmera utilizando a função SendMessage;
- Algumas mensagens da API de Vídeo do Windows e seus respectivos parâmetros;
- Como construir um projeto de detecção de movimentos usando a API de Vídeo do Windows.

Qual a finalidade

- Gravação de vídeo e imagem com equipamentos externos como webcams, câmeras etc.

Quais situações utilizam esses recursos?

- Projetos de aquisição e manipulação de imagens.

Thiago Batista Limeira
(tblimeira@gmail.com)
é engenheiro da computação e trabalha com Delphi há mais de 4 anos, atualmente desenvolve programas e interfaces homem-máquina que envolvam comunicação de dados. Pode ser contatado pelo endereço tblimeira@gmail.com

Atualmente o crescimento de aplicações de vídeo que permitem que imagem e som sejam transportados através da Web tais como programas de bate papo e vídeo conferência, fizeram com que programas como o *Windows Live Messenger* e o *Skype* alcançassem grande popularidade, e câmeras e webcams passaram a ser grandemente utilizadas. Com isto houve o aumento da necessidade de se utilizar os recursos destes incríveis acessórios. Hoje com um pouquinho de conhecimento e a ajuda da *API* de vídeo do *Windows*, é possível desenvolver diversos aplicativos que possuam habilidades de captura de vídeo como, por exemplo: um detector de movimentos, simulação de um ponteiro de mouse com um feixe de luz, leitores de códigos de barras! Sim é possível! E muitas outras. O limite das aplicações de vídeo e imagem está restrito apenas a criatividade.

Por isso, para que esta limitação seja apenas nossas mentes criativas,

demonstrarei como capturar vídeo e imagem através do *Delphi*, utilizando a *AVICAP32.dll* do *Windows* que encapsula todo acesso às funcionalidades de acesso a captura de vídeo e estaremos também desenvolvendo uma aplicação que detectará movimentos. Bom, agora mãos a obra!

A API de Vídeo do Windows

Para desenvolver esta aplicação utilizaremos a *API* de vídeo do *Windows* (*VFW*), dela podemos incorporar facilmente habilidades de captura de vídeo em nossa aplicação usando a classe *AVICap Window*. Esta classe possibilita, mediante utilização de mensagens do *Windows*, acesso ao *hardware* e controle de captura de vídeo. Uma janela criada utilizando a *AVICap Window* possibilita também as seguintes funcionalidades:

- Capturar *streams* de áudio e vídeo em arquivos *AVI*;
- Conectar-se e desconectar-se dinamicamente aos dispositivos de áudio e vídeo;
- Ajustar o *frameskip* de vídeo;
- Mostrar caixas de diálogos para controle da fonte de vídeo e formato;
- Copiar imagens para a área de transferência;
- Salvar imagens como *DIB Bitmaps*.

A **Tabela 1** mostra as mensagens que iremos utilizar em nossa aplicação, juntamente com seus respectivos valores e funções. Não se esqueça de declarar a biblioteca *Controls* que possui a definição da mensagem *WM_USER* e também as mensagens da **Tabela 1** na seção *const* do *form*.

> **Nota**: A constante *WM_USER* que equivale ao valor 1024, é usada pelas aplicações como auxílio para se definir mensagens para uso por classes definidas pelo usuário, normalmente sobre a forma: *WM_USER + X*, onde *X* é um valor inteiro.

Mensagem	Valor	Descrição
WM_CAP_START	WM_USER (1024)	Mensagem Base
WM_CAP_DRIVER_CONNECT	WM_CAP_START +10	Conecta webcam
WM_CAP_DRIVER_DISCONNECT	WM_CAP_START +11	Desconecta webcam
WM_CAP_FILE_SET_CAPTURE_FILEA	WM_CAP_START +20	Define o nome do arquivo AVI
WM_CAP_SAVEDIB	WM_CAP_START +25	Salvar arquivo bitmap
WM_CAP_CONFIGURE	WM_CAP_START +41	Invoca diálogo de configuração
WM_CAP_SET_PREVIEW	WM_CAP_START +50	Mostra imagem da webcam
WM_CAP_SET_PREVIEWRATE	WM_CAP_START +52	Define o tempo em milissegundos, em que a imagem é capturada e mostrada
WM_CAP_SEQUENCE	WM_CAP_START +62	Inicia gravação de vídeo
WM_CAP_STOP	WM_CAP_START +68	Finaliza gravação de vídeo

Tabela 1. Tabela Mensagens Windows Video API

Criando a primeira aplicação exemplo

Iniciaremos agora nossa aplicação, para isso abra o *Delphi* e crie uma nova aplicação. Salve esse projeto como *Webcam.dpr* e o formulário principal como *uMain.pas*. Feito isso poderemos inserir um *Panel* em nosso formulário com a propriedade *Name* alterada para *pnlWebcam*, a *BevelOuter* para *bvNone*, *Caption* para *Webcam não conectada* e *Color* para *clBtnShadow*. Este *panel* será utilizado como nosso container de captura de vídeo. Agora que já temos o *panel* como nossa janela de vídeo, precisamos declarar uma variável do tipo *Hwnd* na seção *private* do nosso *form* principal, conforme segue. Esta variável *HWndCaptura* é um *handle* de janela.

```
HWndCaptura : HWnd;
```

No nosso caso necessitamos deste *handle* para identificar o container de vídeo que utilizaremos, neste exemplo o *pnlWebcam*. Para que possamos manipular uma webcam a partir do *Delphi*, precisamos utilizar a função *capCreateCaptureWindowA* que criará uma janela de captura de vídeo e retornará um *handle* que além de identificar a janela de captura, proporcionará controle sobre a webcam. De acordo com a *API* de vídeo do *Windows* é possível criar uma ou mais janelas de captura e associá-las a diversas câmeras.

Nota do DevMan

Um *handle* é um manipulador, nesse caso de uma janela, ou seja, o *handle* é uma variável de valor inteiro que identifica uma janela ou controle. Toda vez que uma nova janela é criada, o sistema operacional Windows associa um valor ao *handle* a todos os objetos de tela.

Declare a função a seguir acima da seção *implementation*, do *form* principal:

```
function capCreateCaptureWindowA(
  lpszWindowName: PCHAR;
  dwStyle: longint; x: integer; y: integer;
  nWidth: integer; nHeight: integer;
  ParentWin: HWND; nId: integer): HWND;
  stdcall external 'AVICAP32.DLL';
```

Note que esta função pertence a *AVICAP.dll* que encapsula toda a API de vídeo do *Windows*. Insira no *form* 6 botões e modifique suas propriedades *Name* para *btnConectar*, *btnDesconectar*, *btnConfigurar*, *btnGravarImagem*, *btnGravarVideo* e *btnParar*, respectivamente. Coloque também um *SpinEdit* e modifique suas propriedades *Name* para *spnedtRate*, *MinValue* para 1, *MaxValue* e *Value* para 30. Este *SpinEdit* será utilizado para definir a quantidade de *frames* por segundo que a câmera irá captar. Adicione também um componente *Edit* e mude sua propriedade *Name* para *edtFileName* e *Text* para *"C:\Teste.avi"*, e também colocaremos 2 *labels* com os seguintes *Captions*: *Preview Rate* e *Arquivo*. E para finalizar esta etapa coloque no *form* um *CheckBox* e mude sua propriedade *Name* para *chkbxPreview*, com ele poderemos controlar se desejamos visualizar ou não as imagens da câmera. A **Figura 1** apresenta uma sugestão de *GUI* para esta aplicação.

Criando uma janela de captura de vídeo e conectando à webcam

Agora criaremos um *handle* de captura de vídeo e associaremos este *handle* ao nosso *pnlWebcam* que será nosso container, para isso dê um duplo clique no

btnConectar e no evento *OnClick* do botão digite o código da **Listagem 1**.

No clique do botão *btnConectar*, iremos invocar a função *capCreateCaptureWindow*, passando os parâmetros *lpszWindowName* que define o título da janela de captura. O *dwStyle* define o tipo de janela que utilizaremos, os parâmetros *x*, *y*, *nWidth*, *nHeight*, dizem respeito a posição *x*, *y*, largura e altura do nosso *panel*, *ParentWin* é o *handle* do container que utilizaremos e *nID* o identificador da janela que estamos criando. Com isso associamos a nossa variável *HWndCaptura* um *handle* para manipular a *webcam*.

Agora todo o processo de *interfaceamento* da *webcam* se dará por meio das mensagens da *api* de vídeo, para isso utilizamos o comando *SendMessage*, veja detalhes deste método na **Listagem 2.**

Este método se encarrega de enviar estas mensagens para nossa janela de captura, dependendo da mensagem utilizada ela necessitará de um ou mais parâmetros. Repare no código que verificamos: se o *HWndCaptura* é diferente de 0, indica realmente que existe um número associado a ele, em seguida inicializamos a câmera através da mensagem *WM_CAP_DRIVER_CONNECT* que inicia a conexão com a câmera. Agora utilizando a mensagem *WM_CAP_SET_PREVIEWRATE* definimos quantos *frames* por segundo a câmera captará. Esse primeiro parâmetro da mensagem define o tempo em milissegundos em que a imagem da câmera será atualizada, para que possamos variar este valor dividimos 1000 milissegundos (equivalente a 1 segundo) pelo valor da propriedade *Value* do *spnedtRate*, com isso teremos o valor de *frames* por segundo determinado pelo *SpinEdit*. Para finalizar a inicialização da câmera teremos que utilizar a mensagem *WM_CAP_SET_PREVIEW* que permite ativar ou não a visualização das imagens da câmera. Repare no código que se a conexão foi bem sucedida ativamos os demais botões da aplicação e se porventura ocorra algum erro mostramos uma mensagem de aviso.

Listagem 1. Código do evento OnClick do botão de conexão

```
procedure TfrmMain.btnConectarClick(Sender: TObject);
begin
  HWndCaptura := capCreateCaptureWindowA(
    'Windows Video API no Delphi',
    WS_CHILD or WS_VISIBLE, pnlWebCam.Left,   pnlWebCam.Top,
    pnlWebCam.Width, pnlWebCam.Height, pnlWebCam.Handle, 0);

  if HWndCaptura <> 0 then
  begin
    try
      SendMessage(HWndCaptura, WM_CAP_DRIVER_CONNECT, 0, 0);
      SendMessage(HWndCaptura, WM_CAP_SET_PREVIEWRATE, 1000 div    spnedtRate.Value , 0);
      SendMessage(HWndCaptura, WM_CAP_SET_PREVIEW, Ord(chkbPreview.Checked), 0);
      btnDesconectar.Enabled := True;
      btnConectar.Enabled := False;
      btnConfigurar.Enabled := True;
      btnGravarImagem.Enabled := True;
      btnGravarVideo.Enabled := True;
      btnParar.Enabled := True;
    except
      raise;
    end;
end
  else
  begin
    MessageDlg ('Erro ao conectar ao driver da câmera!',mtError, [mbok], 0);
    btnDesconectar.Enabled := False;
    btnConectar.Enabled := True;
    btnGravarImagem.Enabled := False;
    btnGravarVideo.Enabled := False;
    btnParar.Enabled := False;
    end;
end;
```

Listagem 2. Parâmetros do método SendMessage

```
SendMessage(
  { Handle da janela de destino }
  HWND hwnd,
  { Mensagem }
  UINT uMsg,
  { Primeiro parâmetro da mensagem}
  WPARAM wParam,
  { Segundo parâmetro da mensagem }
  LPARAM lParam
);
```

Listagem 3. Código para desconectar a webcam

```
procedure TfrmMain.btnDesconectarClick(Sender: TObject);
begin
  if HWndCaptura <> 0 then
  begin
   SendMessage(HWndCaptura, WM_CAP_DRIVER_DISCONNECT, 0, 0);
   btnDesconectar.Enabled := False;
   btnConectar.Enabled := True;
   btnGravarImagem.Enabled := False;
   btnGravarVideo.Enabled := False;
   btnConfigurar.Enabled := False;
   btnParar.Enabled := False;
  end;
end;
```

Desconectando e configurando a câmera

Para se desconectar da *webcam* devemos utilizar a mensagem *WM_CAP_DRIVER_DISCONNECT*. Para isso dê um duplo clique no *btnDesconectar* e no evento *OnClick* do botão digite o código da **Listagem 3**.

Apenas enviamos a mensagem a API do *Windows* e habilitamos e desabilitamos os controles necessários para o funcionamento da tela. Agora iremos utilizar a mensagem *WM_CAP_CONFI-*

Figura 1. Exemplo de GUI para nossa aplicação demo

GURE, que faz com que uma janela seja mostrada para configuração do formato e tamanho da imagem (**Figura 2**).

Para isto dê um duplo clique no *btnConfigurar* e digite o código a seguir. Veja que novamente apenas enviamos um parâmetro para API que se encarrega de fazer a chamada correta:

```
if HWndCaptura <> 0 then
  SendMessage(HWndCaptura,
    WM_CAP_CONFIGURE, 0, 0);
```

Ativando a visualização da imagem e alterando o número de frames

Vimos que durante a inicialização da câmera utilizamos a mensagem *WM_CAP_SET_PREVIEW* para definir se queremos ativar a visualização da imagem da câmera, agora iremos através do *chkbxPreview* ativar ou desativar a visualização durante o clique do *CheckBox*, para isso no evento *OnClick* do *chkbxPreview* digite o código que segue:

```
if HWndCaptura <> 0 then
  SendMessage(HWndCaptura, WM_CAP_SET_PREVIEW,
    Ord(chkbxPreview.Checked), 0);
```

Com isso toda vez que ocorrer um clique no *CheckBox* iremos enviar novamente a mensagem e atualizar o parâmetro responsável pela ativação com o estado do *CheckBox* utilizando o método *Ord* que converte o valor *boleano* da propriedade *Checked* do componente em um valor inteiro de 0 ou 1.

Agora vamos supor a seguinte situação: digamos que em sua aplicação não exista a necessidade de uma atualização muito rápida das imagens de sua *webcam*, pois além de utilizar muitos recursos do sistema operacional também deixa a aplicação muito lenta, o que fazer então? Podemos aumentar o tempo em que a câmera atualiza as imagens utilizando a mensagem *WM_CAP_SET_PREVIEWRATE* para alterar este tempo de atualização.

Para isso no evento *OnChange* do *spnedtRate* digite o código a seguir. Com isso toda vez que alterarmos o valor do *SpinEdit* atualizaremos a quantidade de *frames* por segundo que desejamos:

```
  if HWndCaptura <> 0 then
    SendMessage(HWndCaptura, WM_CAP_SET_
PREVIEWRATE, 1000 div spnedtRate.Value, 0);
```

Salvando arquivos bitmap e gravando arquivos de vídeo

Já que estamos trabalhando com uma câmera é de se esperar que possamos salvar fotos e arquivos de vídeos, podemos fazer estas tarefas facilmente através das mensagens da *API* de vídeo. Insira no *form* um componente *SaveDialog* que será utilizado para salvarmos arquivos *bitmap* em disco. Para isso dê um duplo clique no *btnGravarImagem* e digite o código da **Listagem 4.**

Repare que no código estamos utilizando a mensagem *WM_CAP_SAVEDIB* para salvar a imagem atual da câmera como arquivo *bitmap* passando como parâmetro a propriedade *FileName* do *SaveDialog* como nome do nosso arquivo *bitmap*. Vamos agora implementar a gravação de vídeo, para isso dê um duplo clique no *btnGravarVideo* e digite o código da **Listagem 5**. Para incluir funcionalidade de vídeo devemos utilizar as mensagens *WM_CAP_FILE_SET_CAPTURE_FILEA* que define o nome do *arquivo* de extensão *AVI* que desejamos gravar, recebendo como parâmetro a propriedade *Text* do *edtFileName* que definimos como o local onde queremos armazenar o arquivo de vídeo. Além desta mensagem também utilizaremos a *WM_CAP_SEQUENCE* que faz com que se inicie o processo de gravação do arquivo de vídeo.

Para parar o processo de gravação existem duas formas: primeiro utilizando a mensagem *WM_CAP_STOP* que envia o comando de parada à *webcam*, para isto no evento *OnClick* do *btnParar* digite o código a seguir, repare que estamos simplesmente enviando esta mensagem para que a gravação seja interrompida:

```
if HWndCaptura <> 0 then
  SendMessage(HWndCaptura, WM_CAP_STOP, 0, 0);
```

Agora veremos o outro método que é bem mais simples. Toda vez que iniciarmos o processo de gravação de vídeo e desejarmos interrompê-lo basta que apertemos a tecla *ESC* do teclado, a tecla irá automaticamente enviar a mensagem de interrupção de gravação para a webcam.

Construindo uma aplicação de detecção de movimento

Agora que já entendemos como funciona a *API* de vídeo utilizando a função *capCreateCaptureWindowA* da *AVICAP32.dll*, que permite a criação de um *handle* para captura de vídeo e imagem, e também utilizamos as mensagens da *API* para enviar comandos a uma câmera ou webcam, iremos agora demonstrar um caso de uso da *API* de vídeo do *Windows*,

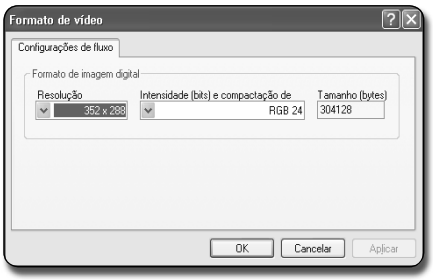


Figura 2. Janela de Configuração da Webcam

Listagem 4. Código para gravação de imagems bitmap

```
procedure TfrmMain.btnGravarImagemClick(Sender: TObject);
begin
  if HWndCaptura <> 0 then
  begin
    SaveDialog1.DefaultExt := 'bmp';
    SaveDialog1.Filter := 'Arquivo Bitmap (*.bmp)|*.bmp';
    if SaveDialog1.Execute then
      SendMessage(HWndCaptura, WM_CAP_SAVEDIB, 0,
        longint(pchar(SaveDialog1.FileName)));
  end;
end;
```

Listagem 5. Código para gravação de vídeo

```
procedure TfrmMain.btnGravarVideoClick(Sender: TObject);
begin
  if HWndCaptura <> 0 then
  begin
    SendMessage(HWndCaptura, WM_CAP_FILE_SET_CAPTURE_FILEA, 0,
      Longint(pchar(edtFileName.Text)));
    SendMessage(HWndCaptura, WM_CAP_SEQUENCE, 0, 0);
  end;
end;
```

em um tipo de aplicação muito comum para monitoramento de ambientes: aquisição de imagens e detecção de movimento!

Existem no mercado vários programas que adquirem imagens de uma câmera de segurança, ou até mesmo de webcams, baseadas em diferença de imagens, ou seja, estes programas apenas armazenam imagens que apresentam alguma alteração ao longo do tempo, isto evita que utilizemos muito espaço da memória em disco e melhora o desempenho do sistema, pois apenas gravamos as imagens que realmente nos interessam, no nosso caso as que possuem alguma diferença de movimento.

Se o caro leitor reparou, no exemplo da aplicação anterior, capturamos imagens *bitmap*, que são maiores e não são compactadas, isto pode ser um problema em um programa de detecção de movimento, pois estaríamos utilizando muita memória ao gravarmos as imagens no formato *bitmap*.

Para solucionar este problema iremos agora implementar uma aplicação de detecção de movimento que gravará imagens no formato *jpg*, que basicamente irá fazer uma varredura entre duas imagens, a atual da câmera e uma imagem anterior. Caso a diferença entre as duas seja maior que um determinado valor, registramos assim movimento e então gravamos a imagem em disco.

Listagem 6. Lista das mensagens do detector de movimento

```
{ Mensagem base }
const WM_CAP_START = WM_USER;
{ Conectar a WebCam }
const WM_CAP_DRIVER_CONNECT = WM_CAP_START + 10;
{ Desconectar a WebCam }
const WM_CAP_DRIVER_DISCONNECT = WM_CAP_START + 11;
{ Preview }
const WM_CAP_SET_PREVIEW = WM_CAP_START + 50;
{ Configurar frames por segundo }
const WM_CAP_SET_PREVIEWRATE = WM_CAP_START + 52;
```

Listagem 7. Código para conexão da webcam e criação de uma janela de captura de vídeo

```
procedure TfrmMain.btnConectarClick(Sender: TObject);
begin
  HWndCaptura := capCreateCaptureWindowA(
    'Detecção de Movimento no Delphi',
    WS_CHILD or WS_VISIBLE,  pnlWebCam.Left,
    pnlWebCam.Top, pnlWebCam.Width,
    pnlWebCam.Height, pnlWebCam.Handle, 0);
  if HWndCaptura <> 0 then
  begin
    try
      SendMessage(HWndCaptura, WM_CAP_DRIVER_CONNECT, 0, 0);
      SendMessage(HWndCaptura, WM_CAP_SET_PREVIEWRATE, 1000, 0);
      SendMessage(HWndCaptura, WM_CAP_SET_PREVIEW, 1, 0);
      btnDesconectar.Enabled := True;
      btnConectar.Enabled := False;
      Timer1.Enabled := True;
    Except
      // Insira um manipulador se desejar
      raise;
    end;
  end
  else
  begin
    MessageDlg('Erro ao conectar ao driver da câmera!',
      mtError, [mbok], 0);
    btnDesconectar.Enabled := False;
    btnConectar.Enabled := True;
    Timer1.Enabled := False;
  end;
end;
```

Listagem 8. Código para desconectar a webcam

```
procedure TfrmMain.btnDesconectarClick(Sender: TObject);
begin
  if HWndCaptura <> 0 then
  begin
    SendMessage(HWndCaptura, WM_CAP_DRIVER_DISCONNECT, 0, 0);
    btnDesconectar.Enabled := False;
    btnConectar.Enabled := True;
  end;
  Timer1.Enabled := False;
end;
```

Criaremos um novo projeto no *Delphi* com o nome de *DetectorMovimento.dpr*, e renomearemos a *unit* como *uMain.pas*, salve o projeto e o arquivo *.pas*. Estaremos nessa aplicação utilizando algumas mensagens do exemplo anterior. Não se esqueça de declarar as mensagens da **Listagem 6** na seção *const* do *form*.

Agora insira um *panel* e renomeie-o para *pnlWebCam*, altere o seu *Caption* para *Webcam não conectada*, e sua propriedade *Color* para *clBtnShadow*, coloque também a propriedade *BevelOuter* para *bvNone*. Insira também três botões e renomeie-os respectivamente como *btnConectar*, *btnDesconectar* e *btnCopiar*. Coloque agora dois componentes *TImage* e altere as suas propriedades *Name* para *imgAtual* e *imgAnterior* respectivamente. E para finalizar insira um *Timer* com sua propriedade *Interval* alterada para 8000. Não se esqueça de declarar a variável *HWndCaptura* do tipo *HWnd* na seção *private* do *form* principal. Dê um duplo clique no *btnConectar* e digite o código da **Listagem 7**.

Se o leitor reparou este código é basicamente o mesmo que utilizamos no exemplo anterior com exceção do *Timer1* que agora também estamos ativando e do intervalo de atualização de imagem da câmera que agora será de 1000, pois atualizaremos a imagem 1 vez por segundo, no caso de detecção de movimento não há necessidade de muita velocidade na atualização de imagens. Dê um duplo clique no *btnDesconectar* e digite o código da **Listagem 8**, que basicamente também é igual ao do nosso primeiro exemplo, só que incluímos o *Timer1* agora.

Neste momento estaremos construindo a rotina de detecção de movimento que utilizaremos em nosso segundo exemplo. Dê um duplo clique no *btnCopiar* e digite o código da **Listagem 9**.

No código da **Listagem 9** ativamos o timer que faz a captura de imagem. Em seguida copiamos a imagem anterior para a propriedade *Picture* do *imgAnterior*. Atualizamos altura e largura da imagem atual bem como sua propriedade *Stretch*. Em seguida invocamos o método *BitBtl* ou *ReleaseDC* conforme o caso. Em seguida começamos a comparar os pixels de cada imagem utilizando dois laços *for*.

Agora chegamos à seguinte pergunta: Como detectar movimento em uma imagem? Comparando uma imagem com outra anterior, por isso precisamos manter uma imagem e comparar os *pixels* individuais que constituem as 2 imagens. Uma imagem de *320x240 pixels* contém *76.800 pixels*. Para cada um dos *pixels* comparamos os valores *Red, Green e Blue (RGB)* com o *pixel* equivalente da outra imagem. Uma forma lenta de se fazer isto é utilizar a propriedade *Pixel* do *Bitmap.Canvas* de ambas as imagens. Esta propriedade *pixel* possui um valor do tipo *TColor*, e para acessar os valores individuais *RGB* eu utilizei um *Record* do tipo *TRGB* conforme segue:

```
type
  TRGB = record
    case Boolean of
      True: (RGBColor: TColor);
      False: (R,G,B: Byte);
  end;
```

Nota: A função *GetDC* recupera o *handle* da área de uma janela específica ou tela inteira. Podemos usar o *Handle* obtido em funções para desenhar no dispositivo de contexto.

Nota: A função *BitBlt* realiza a transferência dos dados de cor de um bloco de *bits* que corresponde a um retângulo de *pixels* de um dispositivo de contexto inicial para um de destino.

O código que faz a comparação dos *pixels* consiste de um *looping* duplo, que compara os valores de *R,G* e *B*. Neste caso subtraímos os valores e verificamos se a diferença é maior do que *$F*. Apesar de funcionar bem na prática, resultados ainda melhores podem ser obtidos checando a diferença relativa entre as duas imagens(dividindo os valores *R,G* e *B* das duas imagens, e certificar-se de que a diferença esteja dentro de 10%, por exemplo). Quando um *pixel* é diferente nas duas imagens, incrementamos a variável *Diff*, como podemos ver no trecho de código a seguir. Note que apesar de que a propriedade *pixel* não seja a maneira mais rápida para acessar esta informação, não é grande problema porque queremos um intervalo de tem-

Listagem 9. Código para captura imagem

```
procedure TfrmMain.btnCopiarClick(Sender: TObject);
type
  TRGB = record
    case Boolean of
      True: (RGBColor: TColor);
      False: (R,G,B: Byte);
  end;
var
  PanelDC: HDC;
  FileName: String;
  Diff,x,y: Integer;
  CurrentRGB,OldRGB: TRGB;

begin
  Timer1.Enabled := False;

  { Copiamos a imagem atual no imgAnterior }
  imgAnterior.Picture.Bitmap := imgAtual.Picture.Bitmap;
  if not Assigned(imgAtual.Picture.Bitmap) then
    imgAtual.Picture.Bitmap := TBitmap.Create
  else
  begin
    imgAtual.Picture.Bitmap.Free;
    imgAtual.picture.Bitmap := TBitmap.Create;
  end;
  imgAtual.Picture.Bitmap.Height := pnlWebcam.Height;
  imgAtual.Picture.Bitmap.Width := pnlWebcam.Width;
  imgAtual.Stretch := True;

  PanelDC := GetDC(Handle);
  try
    BitBlt(imgAtual.Picture.Bitmap.Canvas.Handle,
      0, 0, pnlWebcam.Width, pnlWebcam.Height,
      PanelDC, 0, 0, SRCCOPY);
  finally
    ReleaseDC(Handle, PanelDC);
  end;

  { Comparação das Imagens }
  Diff := 0;
  for x:=1 to pnlWebcam.Width do
  begin
    for y:=1 to pnlWebcam.Height do
    begin
      CurrentRGB.RGBColor :=
        imgAtual.Picture.Bitmap.Canvas.Pixels[x,y];
      OldRGB.RGBColor :=
        imgAnterior.Picture.Bitmap.Canvas.Pixels[x,y];
      if (abs(CurrentRGB.R - OldRGB.R) > $F) and
         (abs(CurrentRGB.B - OldRGB.B) > $F) and
         (abs(CurrentRGB.G - OldRGB.G) > $F) then Inc(Diff);
    end;
  end;

  {Se a diferença em pixels > 600 gravamos a imagem jpg
   usando a data e hora como nome da imagem}
  if Diff > 1000 then
  begin
    with TJpegImage.Create do
    begin
      try
        ShortDateFormat := 'DDmmYYYY';
        ShortTimeFormat := 'HHMMSS';
        Assign(imgAtual.Picture.Bitmap);
        FileName := DateTimeToStr(Now) + '.jpg';
        Caption := FileName + #32 + IntToStr(Diff);
        while Pos(':',FileName) > 0 do
         Delete(FileName, Pos(':', FileName), 1);
        SaveToFile(FileName);
        if Timer1.Interval > 1000 then
          Timer1.Interval := Timer1.Interval div 2;
      finally
        Free;
      end;
    end;
  end
  else
    Timer1.Interval := 8000;

  Timer1.Enabled := True;
end;
```

po antes de atualizarmos a imagem da câmera, é o que faz o trecho de código da **Listagem 10**.

Ao final da detecção de movimento, podemos utilizar o valor da variável *Diff* para determinar se algum movimento foi detectado ou não, e carregar uma nova imagem baseado neste procedimento. Na prática em uma imagem de 320x240, um valor de *Diff* de 1000 ou mais significa um pequeno movimento. Mas pode variar, por exemplo, se colocássemos uma câmera no jardim, o simples fato de uma nuvem encobrir o sol, já é o suficiente para um valor de *Diff* de alguns milhares, mas em um quarto sem movimentação podemos colocar um valor de *Diff* menor.

E outro atributo de nosso detector de movimentos é que para cada foto capturada (em formato Jpg), utilizamos um nome único baseado na data e hora que a foto foi capturada.

E no momento da captura de uma imagem que possui movimento, caso a propriedade *Interval* do *Timer1* seja maior que 1000 (1 segundo) então diminuímos pela metade este valor, para que possamos capturar toda a seqüência de movimentos, do contrário se não houver movimento permanecemos com o valor de *Interval* em 8000 (8 segundos). Não se esqueça de apontar o evento *btnCopiarClick* no evento *OnTimer* do *Timer1* para que o programa funcione corretamente. Então poderemos atualizar a imagem utilizando o clique do botão *Copiar* ou através da interrupção do *Timer1*.

Utilizar detecção de movimento para determinar quando gravar ou atualizar uma imagem pode reduzir significativamente o espaço de disco ou *bandwidth* (para aqueles que desejam transmitir imagens pela internet) que necessitamos. Podemos ver o detector de movimentos em funcionamento na **Figura 3**.

Conclusão

Como pudemos ver neste artigo a *API* de vídeo do *Windows* é muito simples de ser utilizada, e já nos proporciona através de suas mensagens, uma forma rápida de desenvolvimento de aplicações que utilizem câmeras conectadas a computadores. Vimos também como gravar arquivos em formato *jpg* e inclusive implementamos um detector de movimentos simples. Fiquem com Deus, um forte abraço, e até a próxima. ●

Links

MSDN Library - About Video Capture
http://msdn2.microsoft.com/en-us/library/ms704997(VS.85).aspx

Listagem 10. Trecho de código da comparação

```
{Comparação das Imagens}
Diff := 0;
for x:=1 to pnlWebcam.Width do
begin
  for y:=1 to pnlWebcam.Height do
  begin
    CurrentRGB.RGBColor :=
      imgAtual.Picture.Bitmap.Canvas.Pixels[x,y];
    OldRGB.RGBColor :=
      imgAnterior.Picture.Bitmap.Canvas.Pixels[x,y];
    if (abs(CurrentRGB.R - OldRGB.R) > $F) and
       (abs(CurrentRGB.B - OldRGB.B) > $F) and
       (abs(CurrentRGB.G - OldRGB.G) > $F) then Inc(Diff);
  end;
end;
```

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Figura 3. Nosso detector de movimentos em ação

POR QUE A ALOG É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Novidades do Firebird 2.1

Veja o que há de mais interessante na nova versão do banco de dados Firebird

Resumo DevMan

Veremos algumas das principais novidades incluídas na mais nova versão do banco de dados mais utilizado pelos desenvolvedores Delphi: o Firebird.

Nesse artigo veremos

• Novidades do Firebird 2.1.

Qual a finalidade

• Veremos algumas das novidades da nova versão do Firebird.

Quais situações utilizam esses recursos?

• Aplica-se a todos os sistemas baseados em SGBD Firebird.

Adriano Santos

(falecom@adrianosantos.pro.br)

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Gerente de Desenvolvimento SoftPark, empresa parceira Borland. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Um dos bancos de dados mais utilizados pela comunidade *Delphi* teve sua versão *2.1* final lançada em *Abril*. A versão era esperada por muitos devido à grande quantidade de novidades que estavam por serem implementadas. A versão *2.0* já foi uma grande evolução do BD e com certeza muitas das novidades da *2.1* serão muito bem-vindas.

Como costumo dizer, o *Firebird* tornou-se mania entre os desenvolvedores *Delphi* que o usam em larga escala em diversos ramos de atividades. Neste artigo veremos o que há de mais interessante na nova versão. Faremos um *tour* pelas novas funcionalidades e criaremos um banco de dados de exemplo com tudo o que temos de direito.

Veremos as novas compatibilidades com o padrão *ANSI 2003* bem como novas funções que antes só eram possíveis em *UDF's*.

Criando o banco de dados

Para iniciar nosso estudo das novidades do *Firebird 2.1*, vamos primeiramente criar um banco de dados e algumas tabelas necessárias para nossos testes. As tabelas são: *Clientes e Conexoes*. Neste artigo, criaremos o banco de dados utilizando a ferramenta *IBExpert* em sua versão *Standard*, por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de download da ferramenta. Após isso, abra o *IBExpert* e vamos criar o banco de dados usando a opção *Script Executive* presente no menu *Tools*.

Com a tela de *scripts* aberta, digite o código da **Listagem 1**. Nele estão contidos os esquemas para a criação do banco de dados assim como de cada tabela do sistema. Digitado o *script*, basta executá-lo usando o botão *Run Script* ou pressionando a tecla *F9*.

> **Nota:** Substitua *<Caminho>\FIREBIRD 21.FDB* pelo diretório do banco de dados e seu nome, ex: *C:\BancoDeDados\Firebird21.fdb*.

Note que criamos o banco de dados, as tabelas *Clientes* e *Conexoes*, índices e incluímos alguns registros na tabela *Clientes*. Se preferir, observe nas **Figuras 1** e **2** a estrutura de ambas tabelas e na **Figura 3** os clientes incluídos.

Novidades do Firebird 2.1

Pouca gente sabe, mas na pasta de instalação do *FB*, normalmente localizada em *C:\Arquivos de Programas\Firebird\FirebirdXX*, onde *XX* é a versão do banco, podemos encontrar diversos arquivos que descrevem o BD detalhadamente, como por exemplo, o arquivo *Firebird_v2.1.0.ReleaseNotes.pdf* que contém todas as novidades da versão atual, em inglês. Além disso podemos acessar esse arquivo diretamente pelo menu *Iniciar>Programas>Firebird 2.1 (Win32)>Firebird 2.1.0 Release Notes*.

O arquivo de instalação do *FB*, caso ainda não tenha baixado e instalado, encontra-se no site dos desenvolvedores em *www.firebirdsql.org* no link *Download>Firebird Relational Database* e em seguida clicando na imagem *Firebird 2.1*. É possível também efetuar

Listagem 1. Criação do banco e tabelas

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

CREATE DATABASE '127.0.0.1/3050:<Caminho>\Firebird21.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252;

CREATE TABLE CLIENTES (
    ID      INTEGER NOT NULL,
    NOME    VARCHAR(100),
    CIDADE  VARCHAR(50),
    ESTADO  VARCHAR(2)
);

CREATE TABLE CONEXOES (
    ID    INTEGER NOT NULL,
    ACAO  VARCHAR(50)
);

INSERT INTO CLIENTES (ID, NOME, CIDADE, ESTADO) VALUES
  (1, 'ADRIANO SANTOS', 'SÃO PAULO', 'SP');
INSERT INTO CLIENTES (ID, NOME, CIDADE, ESTADO) VALUES
  (2, 'GUINTHER PAULI', 'RIO GRANDE DO SUL', 'RS');
INSERT INTO CLIENTES (ID, NOME, CIDADE, ESTADO) VALUES
  (3, 'GLADSTONE MATOS', 'RIO DE JANEIRO', 'RJ');
INSERT INTO CLIENTES (ID, NOME, CIDADE, ESTADO) VALUES
  (4, 'FABRICIO DESBESSEL', 'RIO GRANDE DO SUL', 'RS');
INSERT INTO CLIENTES (ID, NOME, CIDADE, ESTADO) VALUES
  (5, 'LUCIANO PIMENTA', 'RIO GRANDE DO SUL', 'RS');

COMMIT WORK;

ALTER TABLE CLIENTES ADD CONSTRAINT PK_CLIENTES PRIMARY KEY (ID);
```

Fields

ID INTEGER NOT NULL

#	FK	PK	Field Name	Field Type	Size	Not Null
1		1	ID	INTEGER		☒
2			NOME	VARCHAR	100	☐
3			CIDADE	VARCHAR	50	☐
4			ESTADO	VARCHAR	2	☐

Figura 1. Estrutura da tabela Clientes

Fields

ID INTEGER NOT NULL

#	FK	PK	Field Name	Field Type	Size	Not Null
1			ID	INTEGER		☒
2			ACAO	VARCHAR	50	☐

Figura 2. Estrutura da tabela Conexoes

ID	NOME	CIDADE	ESTADO
1	ADRIANO SANTOS	SÃO PAULO	SP
2	GUINTHER PAULI	RIO GRANDE DO SUL	RS
3	GLADSTONE MATOS	RIO DE JANEIRO	RJ
4	FABRICIO DESBESSEL	RIO GRANDE DO SUL	RS
5	LUCIANO PIMENTA	RIO GRANDE DO SUL	RS

Figura 3. Clientes incluídos na tabela

o download de todas as versões mais antigas do banco caso a versão atual não o contemple. Não entrarei em detalhes quanto à instalação do banco, pois além de não fazer parte do escopo deste artigo, a instalação é muito simples e tranqüila de se fazer.

Muito bem, vejamos então o que há de mais interessante no *FB 2.1* partindo do próprio arquivo de *Release* da versão. Na página *38* do arquivo *PDF*, encontramos os itens a seguir:

- *Database Triggers:* Gatilhos para eventos do banco de dados;
- *Global Temporary Tables:* Tabelas temporárias globais;
- *Column Aliases in CREATE VIEW:* Apelidos para colunas em *Views*;
- *CREATE TRIGGER SQL2003 Variant:* Criação de *Triggers* no formato *SQL 2003*;
- *Alternative Syntax for Computed Fields:* Campos calculados;
- *CREATE SEQUENCE:* O novo esquema de criação de *Generators*;
- *REVOKE ADMIN OPTION:* Revoca, ou seja, retira direitos de administrador de tabelas;
- *SET/DROP DEFAULT Clauses:* Permite *setar*/*dropar* um valor *default* para determinado campo;
- *Syntaxes for Changing Exceptions:* Permite criar, recriar ou alterar uma *Exception*;
- *ALTER EXTERNAL FUNCTION:* Permite modificar o ponto de entrada ou nome do módulo externo, UDF's, quando a mesma não puder ser excluída;
- *COMMENT Statement:* Permite incluir comentários para todo o banco de dados, facilitando a documentação do BD;
- *CREATE VIEW Extensions:* Utilização de algumas funções *DDL* agora em *Views*, como *Order by, Union, Rows* etc;
- *Create FKs Without Exclusive Access:* Criação de chaves estrangeiras sem uso exclusivo, ou seja, dependências;
- *Changed Logic for View Updates:* Permite aplicar regras de *NOT NULL* em colunas da *View*;
- *Descriptive Identifiers for BLOB Subtypes:* Identificadores descritivos de tipos de *Blob*;
- *CREATE COLLATION statement:* Criação de *Collates* no banco de dados.

Infelizmente, por falta de espaço não demonstrarei todas as novidades da versão, até mesmo porque são inúmeras além das que já vimos.

Triggers para eventos do banco de dados e Sequence

Nesse tópico veremos duas novidades do *FB 2.1* de uma única vez. A primeira delas diz respeito ao *Sequence*. O *Sequence* é basicamente a mesma coisa que um *Generator*, ou seja, é um gerador de números. Em nosso caso criaremos uma espécie de *log* em nosso banco de dados na tabela *Conexoes*, como veremos em seguida. Esse *log* terá como gerador seqüencial um *Sequence* com o nome de *Gerador*. A sintaxe para a criação de um *Sequence* é bem simples, veja:

```
CREATE SEQUENCE [NOME_DA_SEQUENCE]
```

Para o nosso exemplo utilize a instrução anterior incluindo o nome de nossa *Sequence*. Mais adiante veremos como utilizá-la em nossos testes.

A segunda grande novidade e uma das mais interessantes é que agora podemos criar *Triggers* para eventos do banco de dados. Dessa forma, podemos criar os mais variados mecanismos em nosso *BD* para as mais variadas situações. Um exemplo típico é a inclusão de uma espécie de *log* de ações. Temos condições de mapear o que ocorre em nosso banco utilizando *Triggers* que podem ser disparadas nos eventos:

- *Connect:* Ao conectar-se no banco de dados;
- *Disconnect:* Ao desconectar-se do *BD*;
- *Transaction Start:* Ao iniciar uma transação;
- *Transaction Commit:* Ao *comitar* – finalizar – uma transação;
- *Transaction Rollback:* Ao cancelar uma transação.

Criamos propositalmente uma tabela *Conexoes* em nosso *BD*. É ela quem será nossa tabela de *log*. Guardaremos nela uma *Sequencia*, como já mencionado, e o nome da *Ação*. Para a criação da *Trigger* usaremos a seguinte sintaxe:

```
CREATE TRIGGER [NOME_DA_TRIGGER]
ON [EVENTO]
  AS
BEGIN
  ...
END
```

Veja que podemos definir o *Nome da Trigger* e em qual evento do banco de dados poderá ser executada. As opções possíveis são as já mencionadas anteriormente, portanto abra uma nova seção de *SQL* usando o menu *Tools>SQL Editor* no *IBExpert* ou pressione *F12*. Crie quatro *Triggers*, uma para cada evento com exceção do *Transaction Rollback*, assim como na **Listagem 2**.

Criadas as *Triggers* é possível notarmos que elas já funcionam perfeitamente. Experimente abrir a tabela *Conexoes* e clicar na *aba Data* no próprio *IBExpert*. Perceba que já temos dados na tabela. A tabela possui dados porque o próprio *IBExpert conecta-se*, *inicia transações* e as *comita* toda vez que fazemos alguma ação, como a criação de *Triggers* por exemplo. Agora experimente desconectar-se do banco e conectar-se novamente. Abra a tabela e veja o resultado. Fiz questão de fazer isso e mostrar na **Figura 4** o resultado do meu banco. Em poucos cliques já tenho mais de *400* registros na tabela.

Como explicado, o *IBExpert*, ou qualquer outra ferramenta de *Front-End* de banco, fará conexões, desconexões, *commits* e *start* de transações toda vez que algo ocorrer, disparando então as *Triggers* do *database* criadas por nós. Procure notar também que o *Sequence* funciona exatamente como um *Generator*.

Para que possamos entender ainda melhor o processo, vamos criar uma pequena aplicação teste. Crie um novo projeto no *Delphi 7.0* ou outra versão de sua preferência usando o menu

ID	ACAO
402	COMITOU TRANSACAO
403	INICIOU TRANSACAO
404	COMITOU TRANSACAO
405	INICIOU TRANSACAO
406	COMITOU TRANSACAO
407	INICIOU TRANSACAO
408	COMITOU TRANSACAO
409	COMITOU TRANSACAO
410	COMITOU TRANSACAO
411	DESCONECTOU DO BANCO
412	CONECTOU AO BANCO
413	INICIOU TRANSACAO
414	COMITOU TRANSACAO
415	INICIOU TRANSACAO

Figura 4. Trecho da tabela Conexões

File>New>Application. Em seguida modifique seu *Caption* para *Triggers de Banco de Dados* e seu *Name* para *frmPrincipal*. Por fim salve a *Unit* com o nome *fPrincipal.pas* e o projeto como *Demo1.dpr*.

Insira nesse formulário um *SQLConnection* (*sqlConexao*) e um *SQLDataSet* (*sdsClientes*) ambos da paleta *dbExpress*. Clique duas vezes no *sqlConexao* e na janela que se abre crie uma nova conexão com o banco de dados criado anteriormente utilizando o *driver Interbase*. Se preferir, observe a **Figura 5** que contém uma configuração básica para acesso ao nosso banco.

> **Nota**: Caso esteja recebendo a mensagem de erro *unavailable database* ao tentar conectar-se ao *FB 2.1* com *driver Interbase*, modifique a propriedade *VendorLib* do *SQLConnection* informando o caminho completo para a *DLL fbclient.dll* do *Firebird 2.1*, normalmente situada em *C:\Arquivos de Programas\Firebird\Firebird_2_1\bin\fbclient.dll*

Agora configure o *sdsClientes* de forma que fique conectado ao *sqlConexao* através da propriedade *SQLConnection*. Insira uma instrução *SQL* em sua propriedade *CommandText* como a seguir:

```
SELECT * FROM CLIENTES
```

Como podemos ver na instrução, apenas selecionamos todos os dados da tabela *Clientes*. Agora o que precisamos é incluir os campos da tabela no *Fields Editor* do componente, para isso clique duas vezes nele e com o botão direito selecione *Add all fields*.

Agora selecione a paleta *Data Access* e arraste um *DataSetProvider*(*dspClientes*), um *ClientDataSet*(*cdsClientes*) e um *DataSource*(*dtsClientes*). Ligue o *cdsClientes* ao *dspClientes* pela propriedade *ProviderName* e o *dtsClientes* ao *cdsClientes* pela propriedade *DataSet*. Adicione os campos da tabela *Clientes* ao *Fields Editor* do *cdsClientes* do mesmo modo que fizemos na *sdsClientes*. Basta agora adicionarmos um *DBNavigator* e um *DBGrid* ligados ao *dtsClientes* pela propriedade *DataSource*. Se preferir clique duas vezes no *DBGrid* e adicione todos os campos clicando no botão *Add all fields*.

Listagem 2. Criação das Triggers de banco de dados

```
CREATE TRIGGER LOG_START
ON TRANSACTION START
AS
BEGIN
  INSERT INTO CONEXOES (ID, ACAO) VALUES
    (NEXT VALUE FOR GERADOR, 'INICIOU TRANSACAO');
END

CREATE TRIGGER LOG_COMMIT
ON TRANSACTION COMMIT
AS
BEGIN
  INSERT INTO CONEXOES (ID, ACAO) VALUES
    (NEXT VALUE FOR GERADOR, 'COMITOU TRANSACAO');
END

CREATE TRIGGER LOG_CONEXAO
ON CONNECT
AS
BEGIN
  INSERT INTO CONEXOES (ID, ACAO) VALUES
    (NEXT VALUE FOR GERADOR, 'CONECTOU AO BANCO');
END

CREATE TRIGGER LOG_DESCONEXAO
ON DISCONNECT
AS
BEGIN
  INSERT INTO CONEXOES (ID, ACAO) VALUES
    (NEXT VALUE FOR GERADOR, 'DESCONECTOU DO BANCO');
END
```

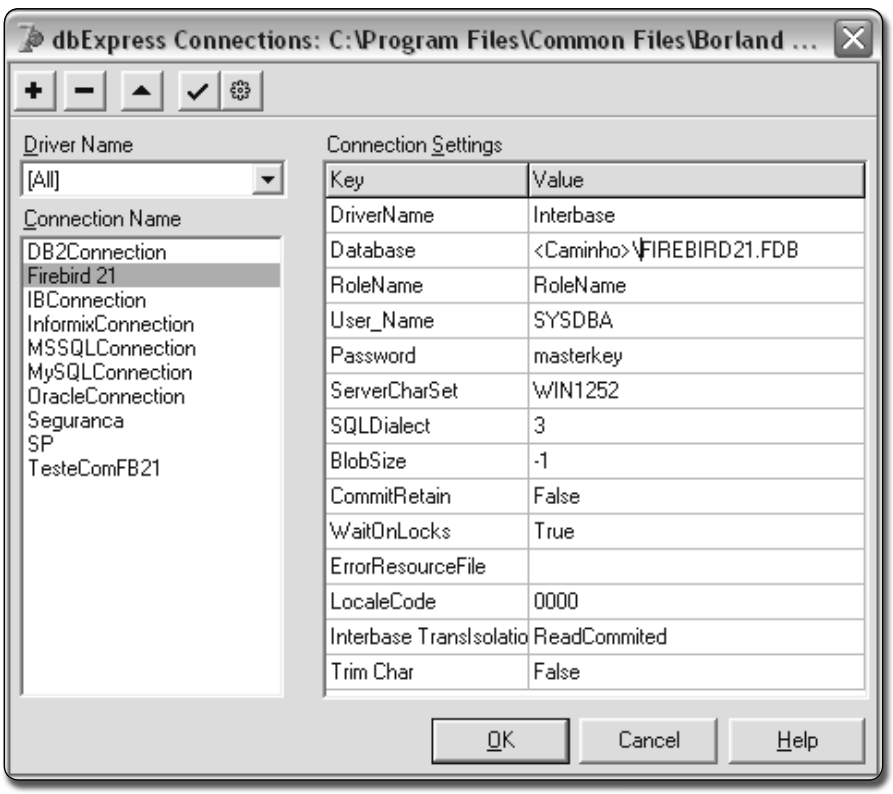


Figura 5. Configuração básica para acesso ao Firebird 2.1

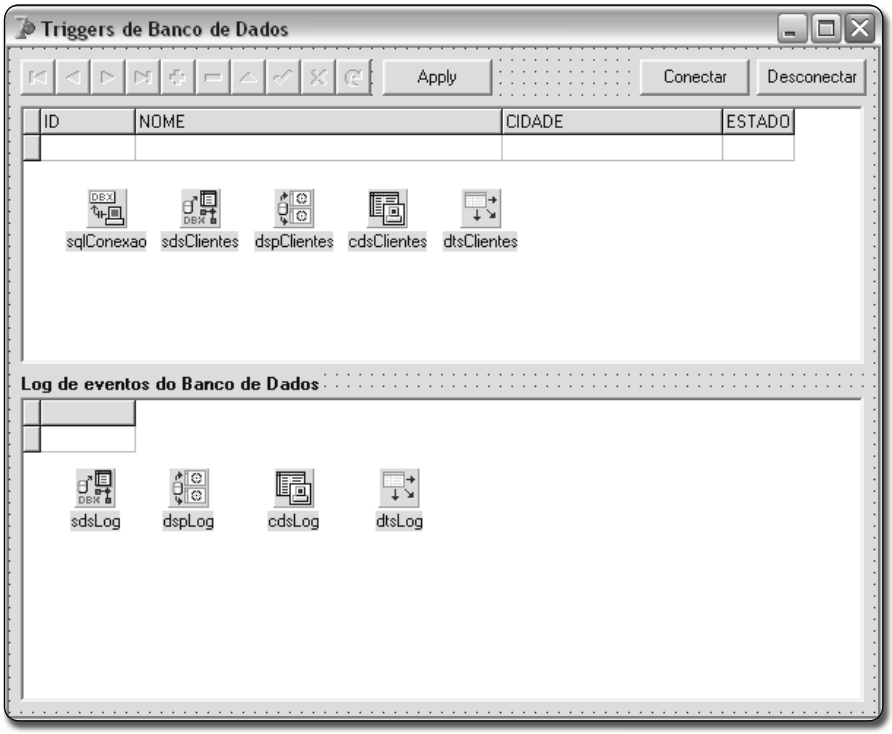


Figura 6. Exemplo de layout para testes com Log

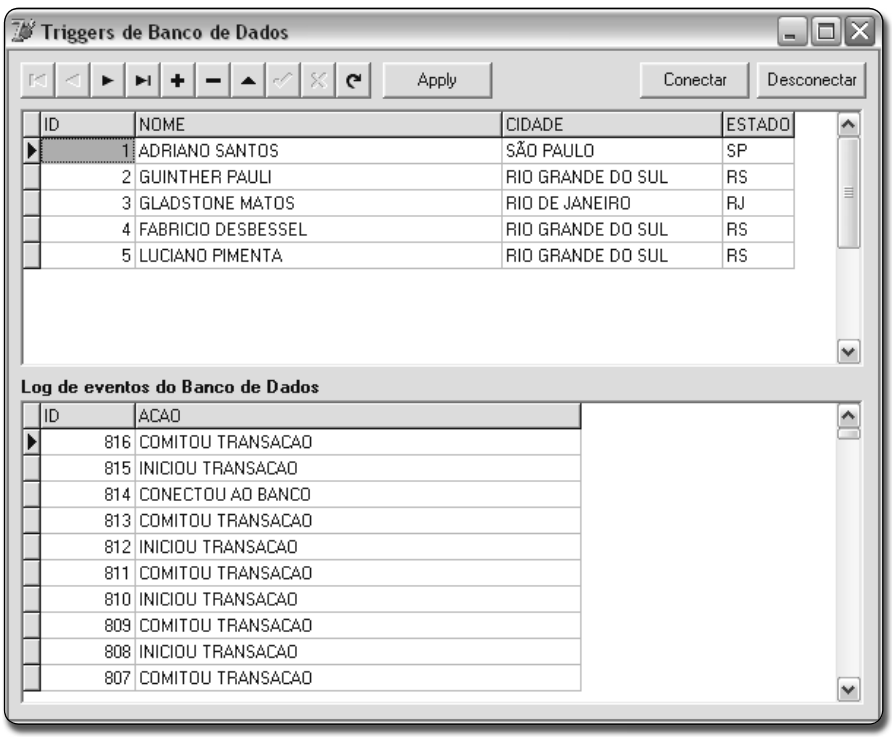


Figura 7. Tabela Conexoes e seus registros incluídos pelas Triggers do banco

Como podemos ver, criamos uma pequena tela de consulta a *Clientes* utilizando *DBGrid*, *DBNavigator* e componentes de conexão a dados usando *DBExpress*. Vamos agora incluir um novo conjunto de componentes *DBExpress* para acesso a tabela *Conexoes*.

Precisamos apenas de um *SQLDataSet* (*sdsLog*), um *DataSetProvider*(*dspLog*), um *ClientDataSet*(*cdsLog*) e um *Data Source*(*dtsLog*), todos ligados entre si, assim como fizemos com *Clientes*. Ligue o *sdsLog* ao *sqlConexao* e inclua a instrução a seguir em sua propriedade *CommandText*.

```
SELECT * FROM CONEXOES ORDER BY ID DESC
```

Novamente, apenas selecionamos os dados da tabela em questão. Dessa vez ordenada pelo *ID* em ordem decrescente para que o maior *ID* fique no topo da lista. Para que possamos ver os dados em tela, arraste um *DBGrid* para o formulário e conecte-o ao *dtsLog*. Veja um exemplo de *layout* na **Figura 6**.

Vamos codificar os eventos *OnCreate* e *OnClose* do formulário de forma que possamos nos conectar ao banco e abrir os dois *ClienteDataSet's* exibindo seus dados. O objetivo disso é mostrar que a tabela *Conexoes* receberá sempre novos registros a cada *conexão, start de transação, commit* e *desconexão*. Modifique os eventos citados conforme a **Listagem 3**.

No evento *OnCreate* do formulário modificamos a propriedade *Connected* do *SQLConnection* para *True* para que possamos conectar-se ao *BD*. Em seguida limpamos os parâmetros dos dois *ClientDataSet* em tela e os abrimos. Nesse ponto nossa *Trigger* de banco já será disparada e um novo registro na tabela *Conexoes* será criado indicando que houve uma conexão.

Já no evento *OnClose*, fazemos o contrário. Fechamos todos os *DataSet's* incluindo os dois *SQLDataSet's*. Por fim desconectamos do banco. Esse processo de desconexão também fará com que uma *Trigger* de nosso *Firebird* seja disparada e um novo registro em *Conexoes* seja criado, dessa vez mostrando que houve uma desconexão.

Claro, ao invés de sempre criarmos um registro na tabela podíamos sim-

plesmente ter apenas alguns registros, localizá-los e incrementá-los à medida que os eventos fossem disparados. Por questões de didática, faremos o teste dessa forma mesmo. Execute o sistema usando a tecla *F9* e ao abrir o programa procure incluir e excluir dados na tabela clientes usando o *DBNavigator*. Feche e abra novamente o sistema. Veja como nossa tabela *Conexoes* ficou (**Figura 7**).

Note que agora temos ainda mais registros, provando que as *Triggers* associadas aos eventos *Connect, Disconnect, Transaction Start* e *Transaction Commit* estão sendo disparadas e funcionam perfeitamente.

Criando Triggers no format SQL 2003

E por falar em *Triggers*, outra grande novidade: agora podemos criar *Triggers* de duas formas diferentes. Usando o formato que já tínhamos antes e agora usando o *SQL 2003*. A antiga sintaxe empregada era:

```
CREATE TRIGGER T1
FOR TABLE
[ACTIVE] BEFORE INSERT OR UPDATE
AS
BEGIN
...
END
```

No formato *SQL 2003*, passamos a informar o nome da tabela depois. Veja:

```
CREATE TRIGGER T2
[ACTIVE] BEFORE INSERT OR UPDATE
ON TABLE
AS
BEGIN
...
END
```

Como podemos ver, informamos em qual tabela aplica-se a *Trigger* só depois do evento, como prevê o *SQL 2003*.

Campo calculados

É difícil encontrar um desenvolvedor que nunca tenha utilizado um campo calculado em seu sistema. Os campos calculados normalmente são utilizados para armazenar, temporariamente, o resultado de alguma fórmula que não necessariamente precisa estar o tempo todo guardado no banco. A exemplo disso podemos citar o total em valores monetários de produtos estocados na empresa. Normalmente multiplica-se o valor do produto pelo preço de mercado dele, totalizando o valor em moeda de determinado produto, tanto para efeitos de estoque quanto para venda. Vejamos um exemplo típico. Abra uma nova seção *SQL* no *IBExpert* e digite o código a seguir:

```
CREATE TABLE PRODUTOS(
  PRODUTO VARCHAR(30),
  QUANTIDADE INT,
  PRECO DOUBLE PRECISION,
  TOTAL GENERATED ALWAYS AS (QUANTIDADE * PRECO)
)
```

O que estamos fazendo no código anterior é criando uma tabela com nome *PRODUTOS* que possui os campos

Produto, Quantidade, Preço e *Total*. Perceba que criamos o campo *Total* de uma maneira bem diferente do que estamos acostumados. Estamos usando o *Generated Always As* que é a sintaxe padrão para criação de campos calculados. Quando fazemos isso com um campo, ele saberá automaticamente que o seu conteúdo é igual ao campo *Quantidade* multiplicado pelo *Preco*. Em nosso projeto Delphi, basta utilizarmos o campo TOTAL diretamente nos controles visuais assim como já fazemos nos demais campos.

Para testar, excute o *script* e clique no botão *Commit Transation* em nossa barra de ferramentas do *IBExpert*. Agora experimente entrar na tabela PRODUTOS e nela inclua alguns registros deixando em branco o campo total. Você notará que ele será automaticamente preenchido (**Figura 8**).

Criando Vews com Aliases

Outro recurso interessante que foi implementado é a criação de *Views* com nomes de campos em forma de *aliases*, ou seja, podemos trocar o nome de um campo da *View*. Veja a sintaxe:

```
CREATE VIEW V_TESTE AS
  SELECT
    ID,
    ID AS CODIGO,
    NOME_DO_FULANO AS NOME
  FROM TAB;
```

Como podemos ver, a sintaxe é bem simples. Apenas montamos nossa *Select* e usamos o operador *AS* entre o campo e o apelido assim como fazemos em *Select's* comuns. Conseqüentemente em *Delphi*, podemos usar o *alias* normalmente, exemplo:

```
SELECT * FROM V_TESTE

ou ainda melhor

SELECT ID, CODIGO, NOME FROM TESTE
```

Criando tabelas temporárias

Um recurso bastante utilizado por programadores em praticamente todas as linguagens de programação, certamente são as tabelas temporárias. Há milhares de maneiras diferentes de fazê-lo. Uma delas é criar um arquivo XML usando um *ClientDataSet* e entrar com os dados em tempo de execução. Outra seria usando o *Paradox*, mas não recomendo nenhuma das duas, pois a nova versão do *Firebird* prevê a criação de tabelas pelo próprio BD.

Podemos criar uma tabela temporária facilmente utilizando a seguinte sintaxe:

```
CREATE GLOBAL TEMPORARY TABLE TEMPORARIA(
  ID INTEGER,
  NOME VARCHAR(100),
  CIDADE VARCHAR(200)
)
```

Veja que a sintaxe é bem simples de se entender. A maior vantagem dessas tabelas temporárias é que elas podem ser alteradas de acordo dois eventos que são:

• *On Commit Preserve Rows:* Preserva o conteúdo dos campos mesmo depois do *Commit* na base de dados;

• *On Commit Delete Rows:* Se por acaso for necessário apagar os registros após o *Commit*.

Considerações finais

Muito bem, essas foram apenas algumas das principais novidades do *Firebird 2.1*. Infelizmente por falta de espaço não foi possível incluir todas as modificações e novidades dessa versão do *Firebird*.

A equipe da *Firebird SQL* tem trabalhado forte para melhorar ainda mais o produto. Tenho informações de que o *Vulcan* está sendo incorporado ao *Firebird* e tudo indica que excelentes novidades virão.

Conclusão

Vimos neste artigo que muitas novidades foram incluídas no *Firebird 2.1*. Certamente cada dia que passa o *FB* torna-se um banco ainda mais estável. Até a próxima e bons códigos. ●

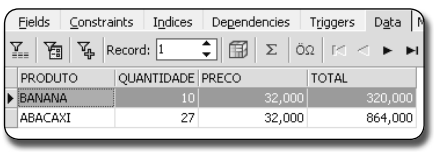


Figura 8. Campo Total sendo atualizado automaticamente

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Renato Matos que mostra as novidades do Firebird 2.1.

*http://www.devmedia.com.br/articles/viewcomp.asp?comp=8799&hl=*firebird*%20and%20*2.1**

Listagem 3. Código dos eventos OnCreate e OnClose do formulário

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  sqlConexao.Connected := True;
  cdsClientes.Params.Clear;
  cdsLog.Params.Clear;

  cdsClientes.FetchParams;
  cdsLog.FetchParams;

  cdsClientes.Open;
  cdsLog.Open;
end;

procedure TForm1.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  cdsClientes.Close;
  cdsLog.Close;
  sdsClientes.Close;
  sdsLog.Close;
  sqlConexao.Connected := False;
end;
```


Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura está acabando!

Renove Já!

www.devmedia.com.br/renovacao

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Pesquisas e Lookups incrementados

Saiba a maneira correta de realizar pesquisas e smart lookups no seu BD

Resumo DevMan

Pesquisas e aplicações cliente/servidor otimizadas fazem uso de queries auxiliares e parâmetros. Eles permitem filtrar corretamente os dados de acordo com a entrada de usuário, localizar um único registro e retornar para a memória. De forma semelhante, smart lookups são soluções inteligentes para permitir a edição de chaves-estrangeiras com base também em queries auxiliares.

Nesse artigo veremos

- Pesquisas automáticas;
- Smart Lookups.

Qual a finalidade

- Permitir a pesquisa correta de registros no BD usando formulários automáticos e reutilizáveis.

Quais situações utilizam esses recursos?

- Toda aplicação cliente/servidor deve fazer uso das técnicas aqui demonstradas.

Neste artigo abordaremos dois importantes fundamentos do desenvolvimento com banco de dados: pesquisas e *lookups*. Nesse contexto, pesquisas referem-se ao fato do usuário abrir um cadastro de clientes, por exemplo, e localizar um cliente para visualização edição. *Lookups* referem-se ao fato do usuário alterar um campo do tipo chave-estrangeira na tabela com base na descrição do campo. Por exemplo, alterar o campo ID_CIDADE do cliente visualizando para isso o nome das cidades, e não seus códigos.

Veremos como implementar ambas as técnicas usando boas práticas de desenvolvimento cliente/servidor, ou seja, usando consultas (*queries*) auxiliares. Componentes como *DBLookupComboBox* só servem para pequenas tabelas, o que não se aplica na maioria dos casos. De praxe, veremos como criar um mecanismo que seja reutilizável, ou seja, que você possa utilizar a pesquisa aqui

Guinther Pauli
(guither@devmedia.com.br)
é Editor geral das Revistas .net Magazine e ClubeDelphi. Certificado Borland Delphi 6, 7, 2005, 2006, Web e Kylix e Microsoft MCP, MCAD, MCSD.NET, MCTS, MCPD. Desenvolve em Delphi desde a versão 1, Turbo Pascal 5.5 e Borland Pascal for Windows.

implementada em todos os cadastros que precisar. Mãos à obra.

Conectando ao BD

Inicie uma nova aplicação Win32 no Delphi e salve todos os arquivos. Crie um novo *DataModule* clicando em *File>New*, dando a ele o nome de *DM*. Salve sua *unit* como *uDM.pas*. Utilizar *DataModules* é uma boa prática pois separa o código de interface do código de acesso a dados, recurso que está presente desde o Delphi 2.

Para este exemplo utilizaremos o *Firebird* como banco de dados e o banco *employee.fdb* como demonstração. Ele contém tabelas que serão úteis e servirão para demonstrar as técnicas aqui apresentadas. Você pode baixá-lo a partir do endereço *www.firebirdsql.org*, no link *Downloads*. A instalação é extremamente simples, de modo que não detalharei aqui. Apenas atente para iniciar o serviço do *Firebird* após a instalação.

Continuando o exemplo, no *DM* coloque um componente *SqlConnection* da guia *dbExpress*. Os componentes desta paleta podem ser utilizados e são otimizados para conexão com o *Firebird*. Dê um duplo clique sobre o componente e clique no botão "+", adicione uma conexão do tipo *Interbase* (que é compatível com o *FB*) e a nomeie como *Employee*. Modifique os parâmetros de conexão conforme mostra a **Figura 1**, atentando para o parâmetro *Database* que deve conter o caminho do seu banco de acordo com a versão do *FB*. Clique em Ok e defina a propriedade *LoginPrompt* do *SqlConnection* para *False*, para evitar que o mesmo solicite o *login* novamente.

Ainda no *SqlConnection*, modifique sua propriedade *VendorLib* de *gds32.dll* para *fbClient.dll*, isso garante que usaremos o *driver* do *Firebird* e não do *Interbase* para conexão. Certifique-se também que *fbclient.dll* seja copiada para a pasta de sistema do *Windows*, ela pode ser encontrada no diretório *bin* de instalação do servidor FB. Teste a conexão configurando a propriedade *Connected* para *True*.

Agora começamos com a parte interessante. Coloque no *DM* um *SqlQuery* (dê o *name* de *qryEmployee*) e aponte sua propriedade *SqlConnection* para *SqlConnection1*. Defina sua propriedade *SQL* para:

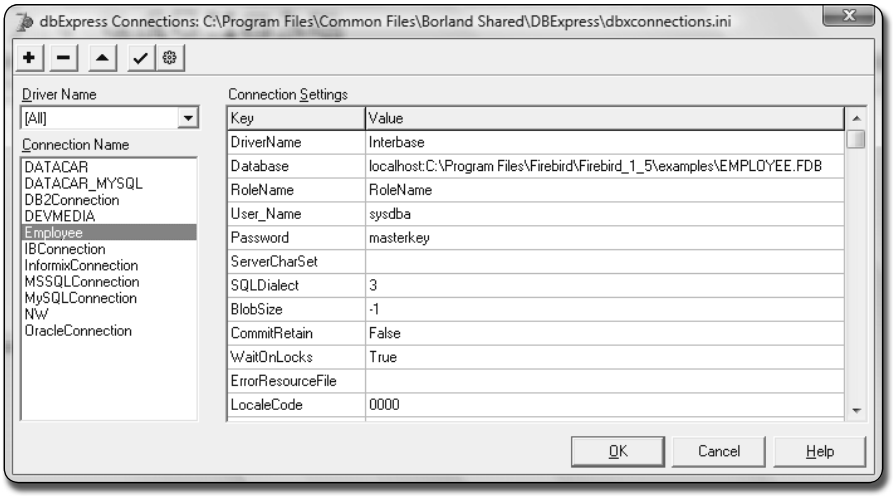


Figura 1. Parâmetros de conexão do Firebird

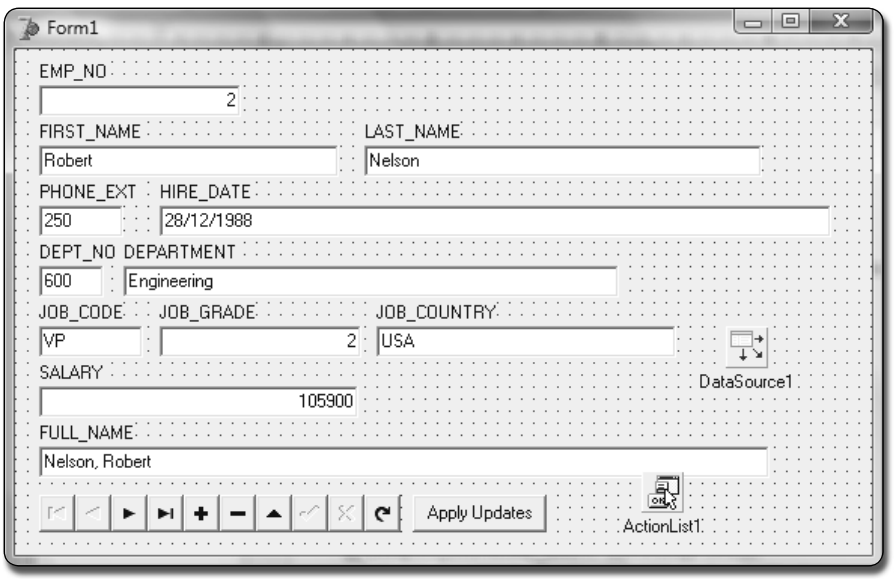


Figura 2. Tela de cadastro de empregados

```
select E.*, D.DEPARTMENT
from EMPLOYEE E left join DEPARTMENT D
on E.DEPT_NO=D.DEPT_NO
```

Nesta instrução estamos trazendo a princípio todos os empregados da tabela *EMPLOYEE*. Nesta tabela, existe um código chamado *DEPT_NO* que é uma referência externa para o departamento do empregado. Nosso usuário não quer saber de códigos, ele precisa ver a descrição desse departamento, por isso fizemos um *join* com a tabela de departamentos, através das chaves *DEPT_NO*. O final é um conjunto de dados com os empregados e seus respectivos departamentos.

Para finalizar, coloque no *DM* um *DataSetProvider* (*dspEmployee)* e um *ClientDataSet (cdsEmployee)*, ambos da guia *Data Access*. Aponte a propriedade *DataSet* do *DataSetProvider* para *qryEmployee* e *ProviderName* do *ClientDataSet* para *dspEmployee*. Esses componentes são necessários para fazer a *cache* local de dados para que possamos trabalhar com *dbExpress*.

Agora dê um duplo clique em *qryEmployee* e adicione todos os campos *TFields*, pressionando *CTRL+A*. Repita a mesma operação para o *cdsEmployee*. Os campos *TFields* nada mais são do que objetos especializados que servem para manipular os campos de uma tabela diretamente.

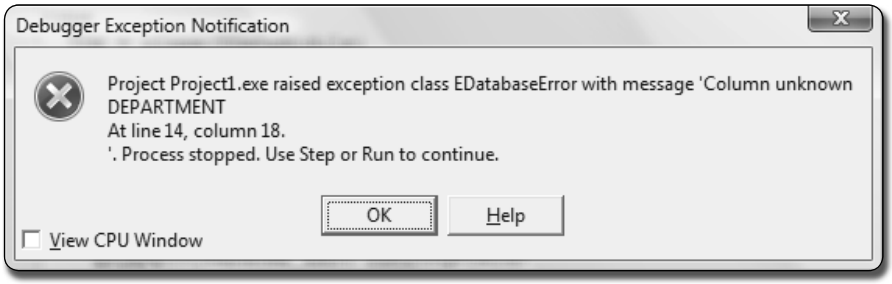


Figura 3. Campo não existe na tabela

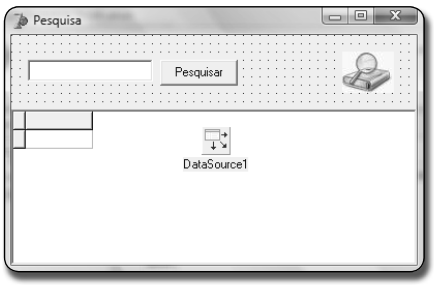


Figura 4. Formulário genérico para pesquisas

Nota do DevMan

Muitos desenvolvedores costumam desabilitar o *Stop On Delphi Exceptions* e acabam não enxergando erros gerados internamente na VCL, como é o caso do *ApplyUpdates* do *ClientDataSet*. Inúmeros desenvolvedores ao chamarem o *Apply* notam que os dados não são efetivados no BD e ficam sem saber a causa. Uma solução nesse caso é chamar o seguinte código no evento *OnReconcileError* do *ClientDataSet*:

```
raise Exception.Create(e.Message);
```

Criando a tela de cadastros

Selecione todos os *TFields* do *ClientDataSet* e arraste-os para dentro da área do formulário. Responda que *Sim* quando o *Delphi* perguntar se quer usar o *DM*. Ajuste os controles em tela ao seu gosto. Coloque um *DBNavigator* da guia *Data Controls* e aponte para o *DataSource* criado através da propriedade de mesmo nome. Esse controle servirá para navegar nos registros.

Coloque um *ActionList* da guia *Standard* e dê um duplo clique sobre o componente. Aperte *Ctrl+Ins* e insira uma ação do tipo *DataSet>TClientDataSetApply*. Volte ao *form*, coloque um *Button* ao lado do *DBNavigator* e na propriedade *Action* aponte para a ação criada. No *DM* configure o *Active* do *cdsEmployee* para *True*. Seu *form* deve estar semelhante ao da **Figura 2.**

Execute a aplicação. Navegue pelos registros e veja que tudo se comporta normalmente. Agora altere o campo *First_Name* e tente gravar e dar um *Apply* no registro. A tela de erro da **Figura 3** aparecerá (certifique-se que *Stop on Delphi Exceptions* esteja ativa em *Tools>Debugger Options>Language Exceptions*). Ela informa que o campo *DEPARTMENT* não existe na tabela que estamos tentando atualizar, *EMPLOYEE*. De fato, esse campo não existe nessa tabela, ele é fruto de um *join*. Ele serve apenas para mostrarmos a descrição do departamento e não o código. Se quiséssemos fazer uma alteração no departamento, precisaríamos atualizar o campo *DEPT_NO*. Veremos como resolver isso mais adiante.

Implementando Pesquisas Automáticas

Nosso primeiro passo rumo a otimização será implementar um mecanismo robusto de pesquisas. *DBNavigators* não existem em aplicações *client/server* reais, pois componentes desse tipo pressupõem que vários registros foram trazidos do servidor. Ora, para que trazer para a memória todos os registros da tabela se o usuário vai visualizar um e alterar outro? Isso é perda de performance. Todo sistema básico *client/Server* tem um mecanismo para permitir que o usuário informe algo que deseja buscar.

Mão na massa. Crie um novo *form*, dê a ele o nome de *frmPesquisa* e salve-o como *uFormPesquisa*. Como você deve ter reparado, o *form* não leva o nome da tabela de empregados, pois esse será um formulário de pesquisa genérico, que poderá ser usado em vários cadastros. Coloque no *form* um *DBGrid* da paleta *Data Controls* (com *Align=alBottom)*, um *Button*, um *Image* e um *Edit*, semelhante à **Figura 4**. Coloque também um *DataSource* e ligue o *DBGrid* a ele.

Agora declare as seguintes variáveis na seção *public* do formulário:

```
public
  //uses SqlExpr, dbClient;
  QrySource: TSQLQuery;
  cdsSource: TClientDataSet;
  cdsAuxiliar: TClientDataSet;
end;
```

Essas variáveis são os parâmetros da pesquisa, ou seja, quais são os *DataSets* fonte que servirão de base para a pesquisa. Eles é que tornam nossa pesquisa genérica. O *QrySource* refere-se a *SqlQuery* de cadastro, nesse caso vai se referir à *qryEmployee*. *cdsSource* é o *ClientDataSet* fonte, nesse caso vamos apontar para *cdsEmployee*. *cdsAuxiliar* é um *ClientDataSet* especial que tem a finalidade somente de pesquisa, e será ligado a componentes auxiliares genéricos que criaremos adiante.

No evento *OnClick* do botão de pesquisa digite o código da **Listagem 1.** Aqui estamos abrindo o *ClientDataSet* auxiliar de pesquisa passando como parâmetro o que foi digitado no *Edit1*. Normalmente vamos utilizar como parâmetro uma instrução *Like*, para trazer os registros com as iniciais digitadas pelo usuário.

Configure a propriedade *Options.dgRowSelect* para *True* do *DBGrid*, para não permitir a edição de registros na pesquisa e selecionar todo o registro quando for clicado. No seu evento *OnDbClick* digite o código da **Listagem 2**. Quando o usuário selecionar um registro, após fazer a pesquisa, e der um duplo clique na janela, nós precisamos configurar a *QrySource* para que traga somente aquele registro selecionado, com base no seu *ID*. Daí estarmos utilizando o primeiro campo do *cdsAuxiliar*, que obrigatoriamente deve ser sempre a chave da tabela.

No *DM*, coloque uma *SqlQuery*(*qryAuxiliar*) e um *DataSetProvider*(*dspAuxiliar*), ligando os componentes pela propriedade *DataSet* do *DataSetProvider* (**Figura 5**). Na propriedade *Options* do *DataSetProvider* habilite a opção *poAllo-*

wCommandText, para que possamos passar as instruções SQL diretamente no *ClientDataSet*. Ambos componentes serão compartilhados por *n ClientDataSets* de pesquisa que criaremos a seguir.

Agora vamos configurar a *query* principal (*qryEmployee*) para que não traga mais todos os registros, mas apenas um, que será disparado através da pesquisa auxiliar. Para isso, altere a sua instrução SQL para:

```
select E.*, D.DEPARTMENT from EMPLOYEE E
left join DEPARTMENT D
on E.DEPT_NO=D.DEPT_NO where EMP_NO=:EMP_NO
```

Observe o parâmetro na chave. É esse parâmetro que é configurado pelo *cdsAuxiliar* na **Listagem 2**. Ou seja, agora estamos realizando boas práticas de programação *client/server*, trazendo apenas o necessário segundo o que o usuário informou. O último passo é configurar o parâmetro criado em *qryEmployee* em sua propriedade *Params*, como mostra a **Figura 6**.

Falta apenas configurar nosso *ClientDataSet* auxiliar que fará a pesquisa propriamente dita. Existe uma regra para que essa pesquisa funcione: a instrução SQL deve retornar como primeiro campo o *ID* da tabela principal onde se está fazendo a pesquisa e o segundo campo deve ser a descrição que receberá o operador *like*.

Largue um *ClientDataSet* no *DM*, ligue-o ao *dspAuxiliar* pela propriedade *ProviderName* e chame-o de *cdsPesqEmployee*. Na sua propriedade *CommandText* digite o seguinte:

```
select EMP_NO, FIRST_NAME from EMPLOYEE
where FIRST_NAME like :FIRST_NAME order by
FIRST_NAME
```

Listagem 1. Clique do botão pesquisar

```
procedure TFrmPesquisa.Button1Click(Sender: TObject);
begin
   DataSource1.DataSet := cdsAuxiliar;
   cdsAuxiliar.Close;
   cdsAuxiliar.Params[0].AsString := Edit1.Text + '%';
   cdsAuxiliar.Open;
end;
```

Listagem 2. Selecionando um registro

```
procedure TFrmPesquisa.DBGrid1DblClick(Sender: TObject);
begin
   QrySource.Params[0].AsInteger := cdsAuxiliar.Fields[0].AsInteger;
   cdsSource.Close;
   cdsSource.Open;
   DataSource1.DataSet.Close;
   Close;
end;
```

Listagem 3. Clique do botão pesquisar

```
//uses uFormPesquisa
procedure TForm1.Button2Click(Sender: TObject);
begin
  FrmPesquisa.QrySource := DM.qryEmployee;
  FrmPesquisa.cdsSource := DM.cdsEmployee;
  FrmPesquisa.cdsAuxiliar := DM.cdsPesqEmployee;
  FrmPesquisa.ShowModal;
end;
```

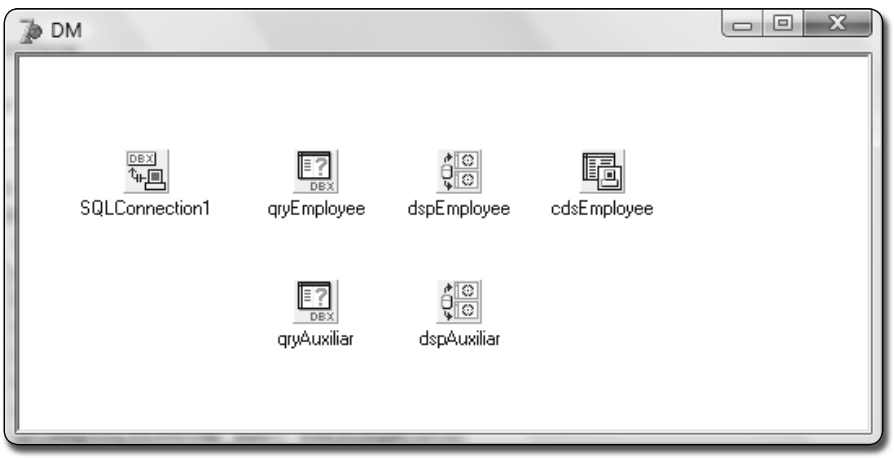


Figura 5. DM com componentes auxiliares

Em sua propriedade *Params* configure o parâmetro criado para *Input* e tipo *string*. Observe que agora você pode ter *n ClientDataSets* auxiliares de pesquisa ligados **ao mesmo** *DataSetProvider*, o *dspAuxiliar*. Para cada *SqlQuery* que necessitar uma pesquisa de abertura, utilize um *ClientDataSet* associado.

Para testar, volte ao *form*, retire o *DBNavigator* e coloque um *Button* (*Pesquisar*). No seu evento *OnClick* digite o código da **Listagem 3**. Aqui estamos configurando os parâmetros do formulário de pesquisa, apontando para os respectivos *DataSets*. E finalmente, exibimos o formulário. O resultado pode ser visto na **Figura 7**.

> **Nota**: Se o formulário de pesquisa não estiver criado, você pode criá-lo manualmente como *Create* e destruí-lo com *Release*.

> **Nota**: Otimizações podem ser facilmente feitas no *DBGrid* de pesquisa. Você pode ocultar o campo chave (que não interessa para o usuário) definindo o *Visible* da primeira coluna como *false* (via código). Os *Captions* nos títulos podem ser ajustados no *ClientDataSet* auxiliar de pesquisa, com base nos campos *TFields*.

O último problema que precisamos resolver para nosso cadastro funcionar foi o mostrado na **Figura 3.** É simples de resolver, no *qryEmployee* do *DM*, acesse os *TFields* e selecione a coluna DEPARTMENT. Configure seus *ProviderFlags* como mostra a **Figura 8**. Aqui estamos dizendo que esse campo não pertence à tabela, não deve ser atualizado, nem faz parte de cláusulas *where*. De forma semelhante, selecione o campo *EMP_NO* e defina seu *pfInkey* para *True*. Mas como atualizar então o campo departamento?

Smart Lookups

Como comentado, *lookups* são formas de editarmos ou vermos chaves-estrangeiras buscando campos em outra tabela relacionada. Comentei que o *DBLookupComboBox* não funciona em situações reais. Então vamos aproveitar nossa estrutura já criada de pesquisas e permitir uma edição de chaves estrangeiras com *smart lookups* (ou *lookups* inteligentes). Ou seja, vamos utilizar o mesmo *form* já criado para implementar esse recurso, só precisamos fazer algumas adaptações.

Abra o *form* de pesquisa e inclua as seguintes variáveis públicas além das que já estavam lá:

```
...
Lookup: boolean;
FieldDesc: TField;
FieldID: TField;
```

O campo *Lookup* vai indicar se a pesquisa é do tipo *Lookup*. O campo *FieldDesc* indica qual é o *TField* responsável

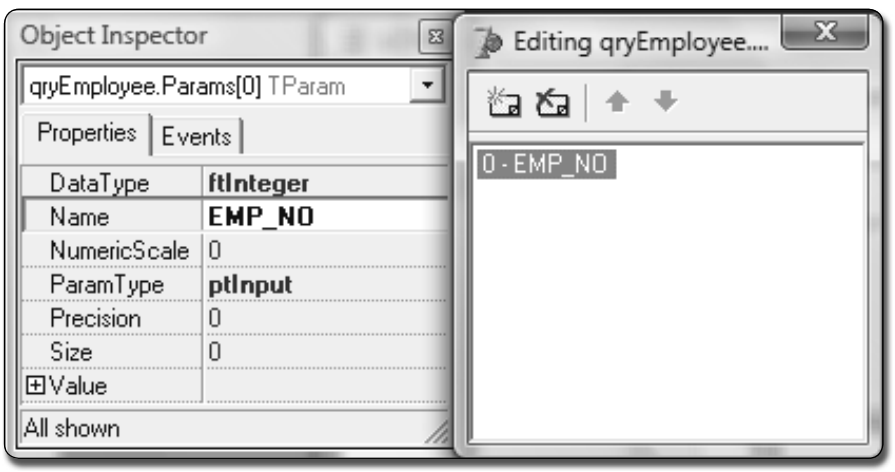


Figura 6. Configurando o parâmetro

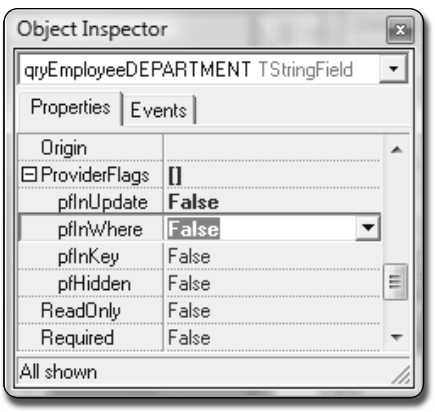


Figura 8. Ajustando os ProviderFlags

Figura 7. Aplicação em execução

Listagem 4. Novo OnDbClick do DBGrid

```
procedure TFrmPesquisa.DBGrid1DblClick(Sender: TObject);
begin
   if Lookup then
   begin
      cdsSource.Edit;
      FieldID.AsInteger := cdsAuxiliar.Fields[0].AsInteger;
      FieldDesc.AsString := cdsAuxiliar.Fields[1].AsString;
   end
   else
   begin
     QrySource.Params[0].AsInteger := cdsAuxiliar.Fields[0].AsInteger;
     cdsSource.Close;
     cdsSource.Open;
   end;
   DataSource1.DataSet.Close;
   Close;
end;
```

Listagem 5. Smart Lookup

```
procedure TForm1.Bit Btn1Click(Sender: TObject);
begin
  FrmPesquisa.Lookup := True;
  //coloque Lookup=false na chamada da pesquisa normal
  FrmPesquisa.FieldDesc := dm.cdsEmployeeDEPARTMENT;
  FrmPesquisa.FieldID := dm.cdsEmployeeDEPT_NO;
  FrmPesquisa.cdsSource := DM.cdsEmployee;
  FrmPesquisa.cdsAuxiliar := DM.cdsPesqDepartment;
  FrmPesquisa.ShowModal;
end;
```

por armazenar a descrição do *lookup* (originalmente o campo que você vê em um *DBLookupComboBox*). *FieldID* nada mais é do que a chave estrangeira que será atualizada após a edição do *lookup*.

No *OnDbClick* do *DBGrid* altere o código como mostrado na **Listagem 4**. Aqui testamos se a pesquisa é do tipo *lookup*, caso seja, atualizamos os campos *TFields* da tabela fonte. Caso contrário, prosseguimos com o procedimento normal de pesquisa.

Agora coloque um *ClientDataSet (cdsPesqDepartment)* no *DM*, vincule-o ao *dspAuxiliar* pela propriedade *ProviderName* (observe que ela usa o mesmo *dsp*) e defina seu *CommandText* para:

```
select DEPT_NO, DEPARTMENT from DEPARTMENT
where DEPARTMENT like :DEPARTMENT
```

Configure o parâmetro (*Params*) como sendo do tipo *Input* e *String*. Agora no formulário principal, retire o campo que edita o código do departamento e coloque um botão, colocando o código da **Listagem 5** no seu evento *OnClick*. Aqui apenas configuramos os parâmetros do formulário de pesquisa, passando os respectivos *TFields* e a *query* de pesquisa *lookup*.

Finalmente, coloque um *Button* no *form* com apenas uma chamada a *DM.cdsEmployee.Post* (como opção você pode retirar a *action* e chamar *Post* e *ApplyUpdates* no mesmo momento).

Conclusão

Vimos que a construção de formulários prontos, sejam eles para pesquisa ou *smart lookups*, podem ajudar e muito na construção de aplicações *Delphi*, pois permitem o reaproveitamento de código e centralização das regras. Pesquisas auxiliares devem ser usadas em ambiente cliente/servidor, pois são mais otimizadas, aliadas ao uso de parâmetros de forma intensiva. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Paginação em Web Sites PHP

Veja como paginar os resultados de suas buscas no banco de dados com PHP

Resumo DevMan

Nesse artigo veremos como instalar e acessar um banco de dados em MySQL, bem como sua criação e manutenção usando o front-end HeidiSQL. Também veremos exibir informações do banco de dados em forma de DBGrid e mostrar um paginador para navegação entre os resultados da busca.

Nesse artigo veremos

• Acesso ao banco de dados MySQL;

• Como criar paginação em páginas em PHP.

Qual a finalidade

• Criar um recurso para navegar entre os resultados de uma busca.

Quais situações utilizam esses recursos?

• A paginação é utilizada sempre que necessitamos navegar entre os resultados de uma determinada busca.

Adriano Santos

(falecom@adrianosantos.pro.br)

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Gerente de Desenvolvimento SoftPark, empresa parceira Borland. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Quando desenvolvemos aplicações *Win32*, normalmente utilizamos o controle *DBGrid* para exibir o resultado de nossas buscas. O *DBGrid* possui grande vantagem em relação a aplicações *Web*, pois consegue trazer nele todo o conteúdo gerado pelo resultado de uma determinada busca ao banco de dados. Além disso, podemos navegar no conjunto de dados para cima e para baixo normalmente. Em conjunto funciona outro controle bastante conhecido da *VCL*(*Visual Component Library*) chamado *DBNavigator*, que por sua vez tem como principal objetivo possibilitar a navegação entre os registros de uma *query*. Juntos, *DBGrid* e *DBNavigator*, formam um par perfeito e mantêm o nível de usabilidade do sistema bastante alto.

A única coisa que ambos componentes não contemplam é justamente mostrar o número de páginas geradas pelo resultado da busca e, conseqüentemente, mostrar links página a página possibilitando o salto diretamente para os registros que mais interessam. Claro, esse recurso é altamente dispensável em aplicações *Win32*, por isso a falta de seu uso.

Já em se tratando de Web, o recurso é interessante e indispensável. Imagine

como faríamos para navegar nas páginas de resultado geradas por uma simples pesquisa no famoso buscador *Google Inc* ou qualquer outro serviço de buscas na internet. Estou falando do recurso de paginação (**Figura 1**). Esse recurso tornou-se indispensável e altamente importante em todo e qualquer *Web Site*.

Neste artigo veremos como criar paginação em nossos *Web Sites* desenvolvidos em PHP. Veremos o quão simples é criar esse recurso e usufruir disso. Mãos na massa!

Instalando e criando o banco de dados

Para iniciar o projeto, vamos primeiramente criar um banco de dados em *MySQL* e uma tabela para que nosso exemplo faça sentido. A tabela será chamada de *Livros* e o *BD* de *Biblioteca*. Neste artigo, criaremos o banco de dados utilizando o a versão *6.0.5 Alpha* do *MySQL* em conjunto com a ferramenta gratuita *HeidSQL 3.2*. Com ela criaremos e administraremos o banco de dados de forma bem tranqüila.

O *MySQL 6.0.1* pode ser encontrado no *http://superdownloads.uol.com.br/download/181/mysql/* lembrando apenas que a versão do *MySQL* disponível é freeware pelo motivo de ser uma versão de testes, porém o *SGBD* se encaixa em licença comercial, ou seja, é necessário adquirir o banco de dados para usufruir de seus recursos. Caso queira ainda é possível trabalhar com *Firebird/Interbase*, *PostGreeSQL* e *Oracle 10g*.

Depois de efetuado o download descompacte o arquivo *mysql-6.0.5-alpha-win32.zip* ou a versão que tenha baixado e em seguida execute o programa *Setup.exe*. Após a tela de boas vindas do *MySQL* selecione a opção típica de instalação e seus componentes. Mantenha as opções padrão, não é necessário configurar nada em específico. Aguarde o término da instalação e a apresentação da tela para configuração do servidor (**Figura 2**).

Mantenha a opção *Configure the MySQL Server now* marcada para que possamos configurar nesse momento o servidor do banco de dados. Clique em *Finish*. Na

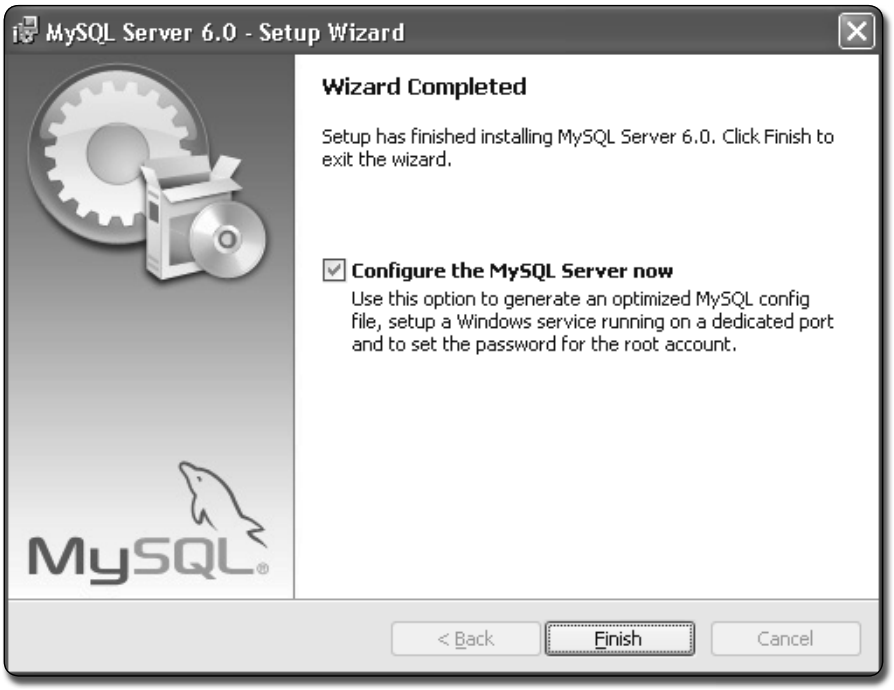


Figura 2. Instalação do servidor

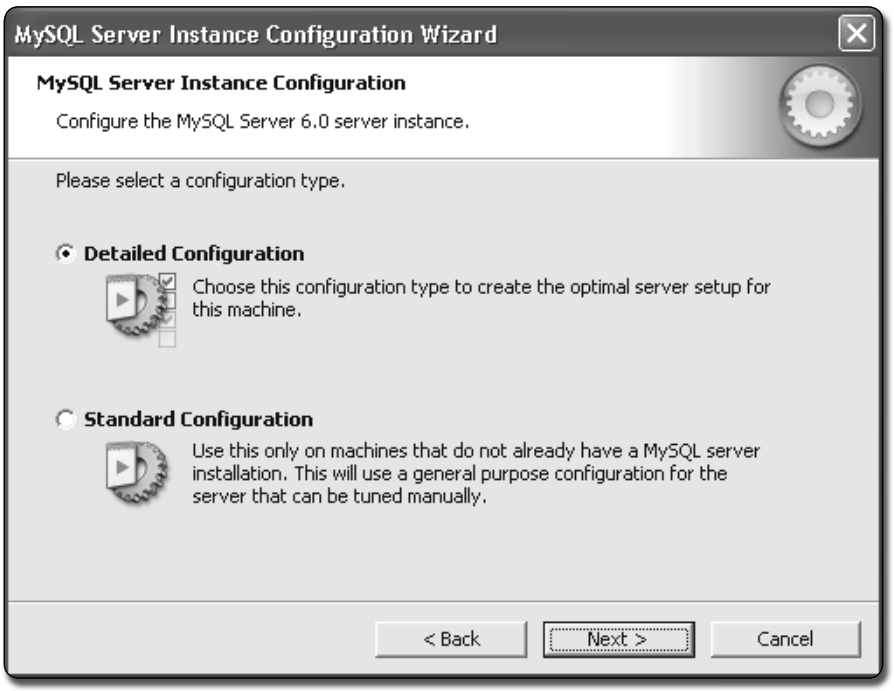


Figura 3. Selecione a opção Configuração Detalhada

Nota do DevMan

O software *HeidSQL 3.2* na verdade é a continuação de um programa que ficou bastante famoso entre os desenvolvedores PHP com banco de dados *MySQL*. O *MySQL Front* foi descontinuado por sua equipe e agora é mantido por outra equipe que renomeou o projeto para *HeidSQL*. A grande vantagem do *HeidSQL* é que é *free* e funciona perfeitamente com qualquer versão do BD, inclusive as mais antigas.

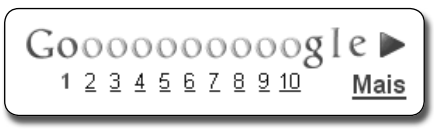


Figura 1. Paginação no Google

tela seguinte marque a opção *Detailed Configuration*. Essa opção é necessária para que possamos configurar todas as opções de instalação do servidor, tais como *porta, usuário* e *senha* (**Figura 3)**.

A próxima tela a ser configurada (**Figura 4**) diz respeito ao modo como instalamos o servidor para testes de desenvolvimento. Use a opção *Developer Machine* para que o *MySQL* utilize menos memória para gerenciamento do *BD*.

Clique em *Next* nas demais telas utilizando o valor default de cada uma, já que não detalharei a configuração de cada uma delas por fugir do escopo desse artigo. Pare quando o banco solicitar *senha* do *root*, o *super-usuário* do *MySQL* (**Figura 5**).

Nota: O usuário padrão do *MySQL* é *root*, por isso em sua instalação o assistente pede apenas a senha. Portanto, em resumo, quando precisarmos nos conectar ao banco de dados digitaremos sempre *root* como usuário e a senha que informamos na instalação. Através do gerenciador do banco é possível incluirmos outros usuários bem como determinar as regras de acesso de cada um.

Na tela final clique em *Finish* para que o *MySQL* efetue as tarefas que estão descritas na tela, incluindo o *start* do servidor. Veja na **Figura 6** que o *MySQL* foi instalado com sucesso e todos os seus serviços foram instalados e configurados. Apenas clique em *Finish*.

Pronto, agora necessitamos fazer a instalação da ferramenta de *Front-End* do banco. Acesse o link *www.heidisql.com* e clique no link *Download*. Você será redirecionado para o site da *SourceForge* e logo em seguida o download iniciará. Depois de concluído o download, basta instalar a ferramenta que não requer configurações adicionais, ou seja, basta seguir a instalação clicando em *Next* em todas as telas (**Figura 7**).

Após a instalação do *HeidiSQL* abra o aplicativo e veja que somos obrigados a cadastrar um host válido (**Figura 8**). Preencha os dados com usuário e senha e clique em *Connect*, após isso uma tela como a **Figura 9** será exibida.

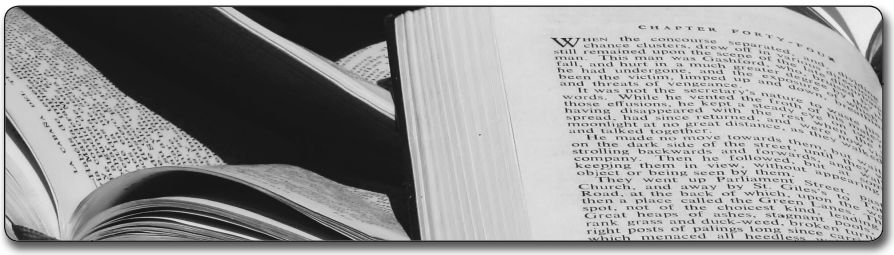
Faremos a criação do banco de dados. Clique com o botão direito do mouse

Figura 4. Marque a primeira opção para que o MySQL utilize menos memória

Figura 5. Digitação da senha do banco de dados

Figura 6. Final da instalação do MySQL

Figura 7. Início da instalação do HeidSQL



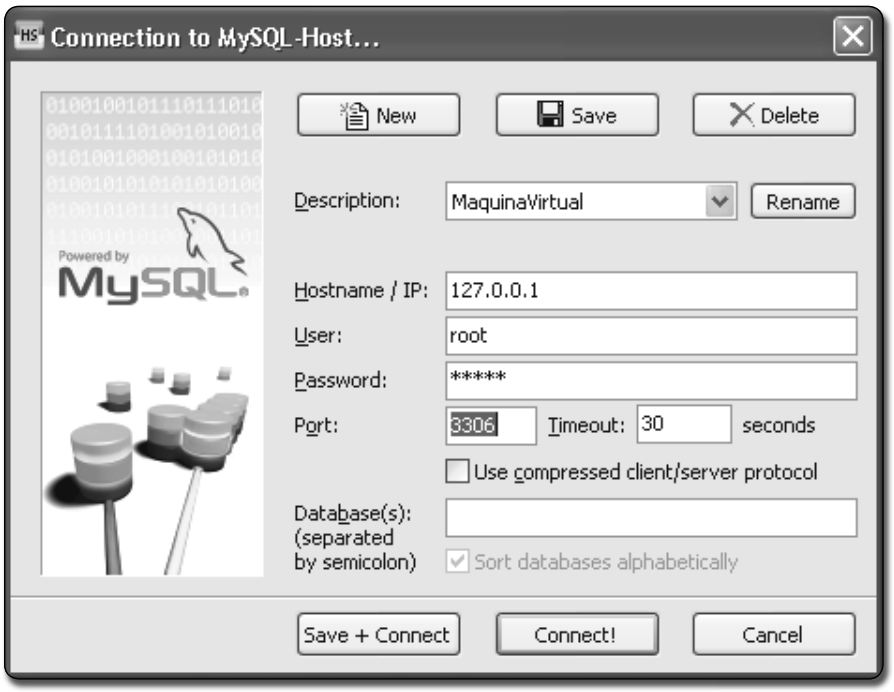


Figura 8. Cadastramento do host

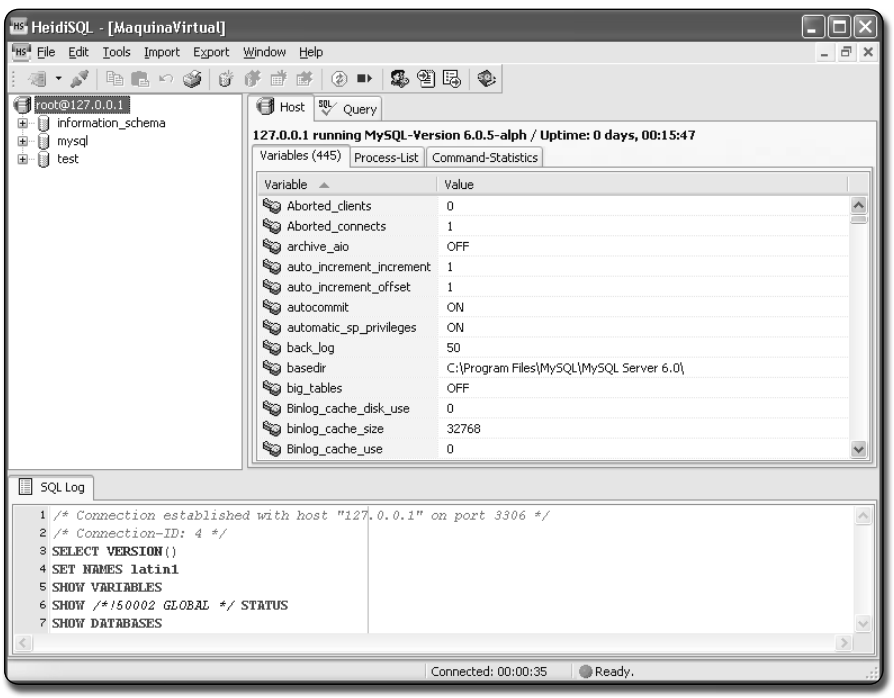


Figura 9. Tela principal do aplicativo HeidiSQL

sobre o nó pai (*127.0.0.1*) na árvore de itens a esquerda do programa. Clique agora em *Create DataBase* e em seguida dê o nome do banco no campo *Name* (**Figura 10**).

Para criar o banco, faremos uso da tela de *scripts SQL*. Clique na *aba Query* e digite o código da **Listagem 1**. Digitado o *script*, basta executá-lo usando o botão *Execute SQL (F9)* ou pressione a tecla *F9*. Se preferir observe as **Figuras 11** e **12** que possuem a estrutura e registros pré-cadastrados do banco de dados.

Criando a aplicação

A primeira coisa que devemos ter em mente para iniciar nosso projeto é: como faremos a exibição somente dos itens que devem ser impressos na página? Bem, primeiro vamos ver como nossa aplicação deverá se comportar.

Quando o internauta fizer uma consulta ao nosso banco de dados, ou seja, abrir o arquivo *index.php* que criaremos a seguir, a página deverá selecionar apenas a quantidade de registros que é permitida por página. A regra serão 10 registros por página. A primeira etapa consiste em verificar o número máximo de registros que existem na tabela e determinar o máximo por página. Comece digitando o código da **Listagem 2**.

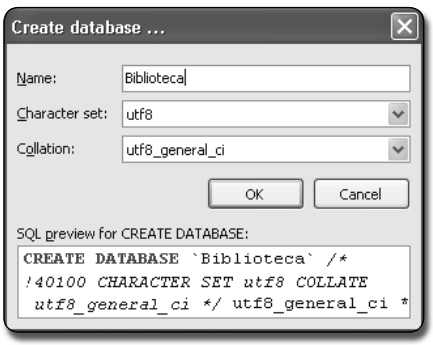


Figura 10. Criação do banco de dados

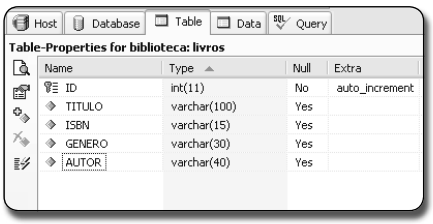


Figura 11. Estrutura da tabela

Listagem 1. Criação do banco e tabelas

```
CREATE TABLE LIVROS (
    ID      INTEGER NOT NULL AUTO_INCREMENT,
    TITULO  VARCHAR(100),
    ISBN    VARCHAR(15),
    AUTOR   VARCHAR(40),
    GENERO  VARCHAR(30)
);

INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (1, 'ANJOS E DEMÔNIOS', '122522336411246', 'DAN BROWN', 'SUSPENSE');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (2, 'FORTALEZA DIGITAL', '451151613163413', 'DAN BROWN', 'SUSPENSE');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (3, 'CÓDIGO DA VINCI', '545346564651313', 'DAN BROWN', 'SUSPENSE');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (4, 'A ARTE DA GUERRA', '544651565464465', 'JULIETA ALÍCIA', 'GUERRA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (5, 'O SEGREDO DAS ÁGUAS', '564321681316445', 'MARIETA BACELAR', 'AVENTURA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (6, 'DELPHI 2007', '357431374373423', 'ADRIANO SANTOS', 'INFORMÁTICA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (7, 'CERTIFICAÇÃO DE A a Z', '231513516546546', 'RUBICAN ULYSSES', 'INFORMÁTICA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (8, 'ELEMENTO X', '324653351651331', 'PAOLA MINERVA', 'AVENTURA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (9, 'SAUDAÇÕES MRS. X', '534646000056061', 'WALTER MUNOZ', 'AVENTURA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (10, 'MACACOS ME MORDAM', '864654546131138', 'ALBERT EISTEN', 'AVENTURA');
INSERT INTO LIVROS (ID, TITULO, ISBN, AUTOR, GENERO) VALUES
  (11, 'DISCOVERY', '343461651666651', 'DISCOVERY', 'ENTRETENIMENTO');
```

Listagem 2. Início do código, descobrindo quantos registros a tabela tem

```
<?php
  $username = 'root';
  $password = 'admin';

  //Conectamos no host, selecionamos ao banco e executamos a query
  $conexao = mysql_connect('localhost', $username, $password);
  mysql_select_db('biblioteca');
  $sql = "select * from livros";
  $resultado = mysql_query($sql);

  //Detectamos a quantide de registros na tabela e o númermo
  //máximo por página
  $recordcount = mysql_num_rows($resultado);
  $total_registros = 10;

  //Determinando o número de páginas que existirão
  if ($recordcount <= $total_registros) {
    $total_paginas = 1;
  }
  if ($recordcount%$total_registros == 0){
    $total_paginas = $recordcount / $total_registros;
  }
  else{
    $total_paginas = ($recordcount/$total_registros) -
      (($recordcount%$total_registros)/$total_registros) + 1;
  }

  //Desconectamos e limpamos o resultado
  mysql_close($conexao);
  mysql_free_result($resultado);

?>
```

	ID	TITULO	ISBN	AUTOR	GENERO
	1	ANJOS E DEMÔNIOS	122522336411246	DAN BROWN	SUSPENSE
	2	FORTALEZA DIGITAL	451151613163413	DAN BROWN	SUSPENSE
	3	CÓDIGO DA VINCI	545346564651313	DAN BROWN	SUSPENSE
	4	A ARTE DA GUERRA	544651565464465	JULIETA ALÍCIA	GUERRA
	5	O SEGREDO DAS ÁGUAS	564321681316445	MARIETA BACELAR	AVENTURA
	6	DELPHI 2007	357431374373423	ADRIANO SANTOS	INFORMÁTICA
	7	CERTIFICAÇÃO DE A a Z	231513516546546	RUBICAN ULYSSES	INFORMÁTICA
	8	ELEMENTO X	324653351651331	PAOLA MINERVA	AVENTURA
	9	SAUDAÇÕES MRS. X	534646000056061	WALTER MUNOZ	AVENTURA
	10	MACACOS ME MORDAM	864654546131138	ALBERT EISTEN	AVENTURA
▶	11	DISCOVERY	343461651666651	DISCOVERY	ENTRETENIMENTO

Figura 12. Livros pré-cadastrados no banco via script

Na **Listagem 2** estamos determinando logo de início que o usuário e senha do banco são *root* e *admin*, respectivamente. Note que as variáveis *$username* e *$password* são usadas posteriormente no método *mysql_connect()* que por sua vez conecta-se ao *host*. Logo em seguida, usamos os métodos *mysql_select_db()* e *mysql_query()* para selecionar o banco no host conectado e executar a instrução *SQL* passada, respectivamente.

> **Nota:** Além do host local, podemos nos conectar também em outros servidores do *MySQL*, seja local ou remotamente. O protocolo utilizado para esse fim é o TCP/IP, por isso a facilidade. Para isso basta entrar no menu *File>Connection* e criar uma nova conexão indicando um novo endereço de IP para que o *HeidSQL 3.2* consiga localizar o servidor.

Nas duas próximas linhas criaremos uma variável chamada *$recordcount* para contar a quantidade total de registros existentes em nossa tabela. A variável *$total_registros* já é auto-explicativa. Sua função é determinar quantos registros caberão em cada página, ou melhor, quantos serão impressos.

Em seguida vem a parte mais interessante. Com base no *$recordcount* e *$total_registros*, determinamos quantas páginas haverá. Vejamos: se o número de registros no total (*$recordcount*) for menor que *$total_registros*, significa que temos menos que 10 registros a serem impressos, então o número de páginas é 1. Agora caso o *$recordcount* dividido por *$total_registros* retorne um número 0, significa que são múltimos de 10, ou seja, dividimos um pelo outro e adquirimos o número exato de páginas. Por fim o último, calculamos quantos registros faltam para preencher a última página, caso a divisão de ambos não seja exata.

Então fechamos a conexão e liberamos a variável de resultado. Até aqui conseguimos saber o número de páginas que o *script* irá gerar.

O trecho a seguir tem uma função importante na página. Criamos uma variável *$pagina* que será enviada na *URL* do arquivo *index.php*, ou seja, quando clicarmos em um dos links em nosso navegador de páginas, um número será enviado para *index.php* e nossa página será recarregada. Automaticamente detectamos através do método *$_GET*, se um valor está vindo na variável. Caso haja valor ele será subtraído de 1 e multiplicado por *$total_registros*. Isso é necessário para que possamos fazer um novo *select* no banco e trazer somente os registros que nos interessam.

```
$pagina = $_GET["pagina"];
if (!$pagina){
  $inicio = 0;
  $pagina=1;
}else{$inicio =
  ($pagina - 1) * $total_registros;}
```

Feito isso, conectamos novamente no banco de dados e executamos uma *query* na tabela. Porém dessa vez trazemos somente os registros necessários. Veja:

```
$conexao = mysql_connect('localhost',
  $username, $password);
mysql_select_db('biblioteca');
$sql = "select * from livros limit " .
  $inicio . ", " . $total_registros;
$resultado = mysql_query($sql);
```

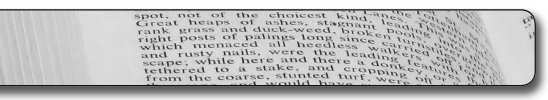
Veja nossa instrução *SQL*. Usamos a cláusula *LIMIT*, que é uma função do próprio *MySQL*. Concatenamos então na *string* da instrução *SQL* as variáveis *$inicio* e *$total_registros*. Dessa forma teremos algo mais ou menos assim em execução:

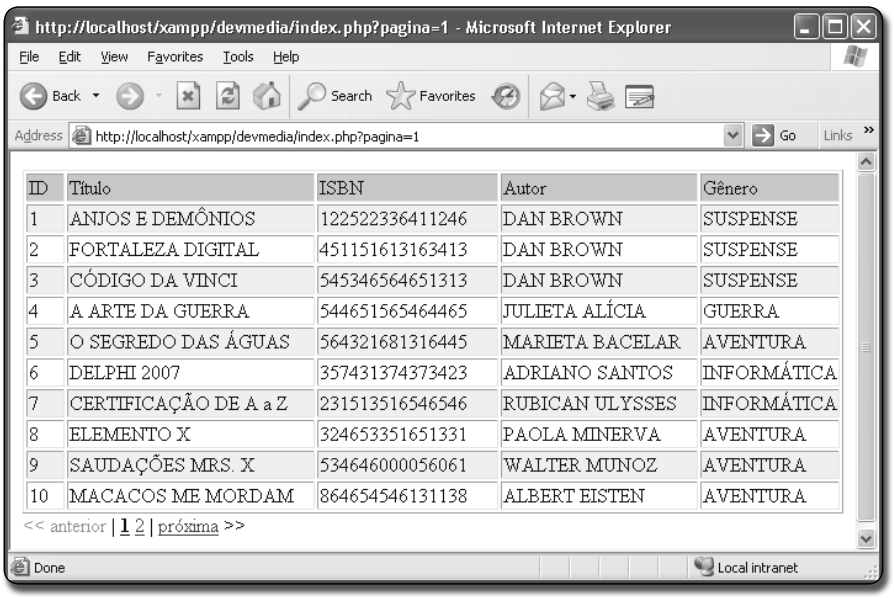
```
select * from LIVROS limit 0, 10
```

No exemplo anterior selecionamos os 10 primeiros registros da tabela. O primeiro parâmetro será sempre variável, pois podemos ter, por exemplo, a expressão a seguir indicando que estamos carregando do registro 258 mais 10 registros:

```
select * from LIVROS limit 258, 10
```

Continuando nosso artigo, agora pulamos para a **Listagem 3.** Nesse momento imprimimos os dados da tabela em uma *TAG <table>*, ou seja, *HTML* puro, sem segredos. A única coisa que devemos nos atentar é que usamos um *array* para





ID	Título	ISBN	Autor	Gênero
1	ANJOS E DEMÔNIOS	122522336411246	DAN BROWN	SUSPENSE
2	FORTALEZA DIGITAL	451151613163413	DAN BROWN	SUSPENSE
3	CÓDIGO DA VINCI	545346564651313	DAN BROWN	SUSPENSE
4	A ARTE DA GUERRA	544651565464465	JULIETA ALÍCIA	GUERRA
5	O SEGREDO DAS ÁGUAS	564321681316445	MARIETA BACELAR	AVENTURA
6	DELPHI 2007	357431374373423	ADRIANO SANTOS	INFORMÁTICA
7	CERTIFICAÇÃO DE A a Z	231513516546546	RUBICAN ULYSSES	INFORMÁTICA
8	ELEMENTO X	324653351651331	PAOLA MINERVA	AVENTURA
9	SAUDAÇÕES MRS. X	534646000056061	WALTER MUNOZ	AVENTURA
10	MACACOS ME MORDAM	864654546131138	ALBERT EISTEN	AVENTURA

Figura 13. Sistema funcionando com paginação

Listagem 3. Impressão dos itens em página

```
$i = 1;
print "<table border='1'>";
print "  <tr>";
print "    <td width='030'bgcolor='#CCCCCC'>ID</td>";
print "    <td width='220'bgcolor='#CCCCCC'>Título</td>";
print "    <td width='150'bgcolor='#CCCCCC'>ISBN</td>";
print "    <td width='180'bgcolor='#CCCCCC'>Autor</td>";
print "    <td width='100'bgcolor='#CCCCCC'>Gênero</td>";
print "  </tr>";

while ($row = mysql_fetch_array($resultado)) {
  if ($i % 2 == 0){
    $cor = "#FFFFFF";
  }else{
    $cor = "#F2F2F2";
  }

  print "<tr>";
  print "  <td bgcolor=$cor>" . $row['ID'] . "</td>";
  print "  <td bgcolor=$cor>" . $row['TITULO'] . "</td>";
  print "  <td bgcolor=$cor>" . $row['ISBN'] . "</td>";
  print "  <td bgcolor=$cor>" . $row['AUTOR'] . "</td>";
  print "  <td bgcolor=$cor>" . $row['GENERO'] . "</td>";
  print "</tr>";

  $i = $i + 1;
  $marcador = $marcador + 1;
}

print "</table>";
```

Listagem 4. Impressão do navegador de links

```
if ($pagina == 1) {
  print "<font color=\"#999999\"><< anterior</font> |";
}else{
  print "<< <a href=\"index.php?pagina=".
    ($pagina - 1)."\" targe=\"_self\">anterior</a> |";
}
$x = 1;
while ($x <= $total_paginas) {
  if ($x == $pagina){
    print " <b><u>$x</u></b> ";
  }else{
    print " <a href=\"index.php?pagina=".$x.
     "\" target=\"_self\">".$x."</a> ";
  }
  $x = $x + 1;
}
if ($pagina == $total_paginas){
  print "| <font color=\"#999999\">próxima >></font>\n";
}else{
  print "| <a href=\"index.php?pagina=".
    ($pagina + 1)."\" targe=\"_self\">próxima</a> >>\n";
}
```

Listagem 5. Código completo da página de resultados

```
<?
  $username = 'root';
  $password = 'admin';
  $conexao = mysql_connect('localhost', $username, $password);
  mysql_select_db('biblioteca');
  $sql = "select * from livros";
  $resultado = mysql_query($sql);
  $recordcount = mysql_num_rows($resultado);
  $total_registros = 10;
  if ($recordcount <= $total_registros) {
    $total_paginas = 1;
  }
  if ($recordcount%$total_registros == 0){
    $total_paginas = $recordcount / $total_registros;
  }
  else{
    $total_paginas = ($recordcount/$total_registros) -
      (($recordcount%$total_registros)/$total_registros) + 1;
  }

  mysql_close($conexao);
  mysql_free_result($resultado);

  $pagina = $_GET["pagina"];
  if (!$pagina){
    $inicio = 0;
    $pagina=1;
  }else{
    $inicio = ($pagina - 1) * $total_registros;
  }

  $conexao = mysql_connect('localhost', $username, $password);
  mysql_select_db('biblioteca');
  $sql = "select * from livros limit " . $inicio . ", " . $total_registros;
  $resultado = mysql_query($sql);

  $i = 1;
  print "<table border='1'>";
  print "  <tr>";
  print "    <td width='030'bgcolor='#CCCCCC'>ID</td>";
  print "    <td width='220'bgcolor='#CCCCCC'>Título</td>";
  print "    <td width='150'bgcolor='#CCCCCC'>ISBN</td>";
  print "    <td width='180'bgcolor='#CCCCCC'>Autor</td>";
  print "    <td width='100'bgcolor='#CCCCCC'>Gênero</td>";
  print "  </tr>";

  while ($row = mysql_fetch_array($resultado)) {
    if ($i % 2 == 0){
      $cor = "#FFFFFF";
    }else{
      $cor = "#F2F2F2";
    }

    print "<tr>";
    print "  <td bgcolor=$cor>" . $row['ID'] . "</td>";
    print "  <td bgcolor=$cor>" . $row['TITULO'] . "</td>";
    print "  <td bgcolor=$cor>" . $row['ISBN'] . "</td>";
    print "  <td bgcolor=$cor>" . $row['AUTOR'] . "</td>";
    print "  <td bgcolor=$cor>" . $row['GENERO'] . "</td>";
    print "</tr>";
    $i = $i + 1;
    $marcador = $marcador + 1;
  }

  print "</table>";

  if ($pagina == 1) {
    print "<font color=\"#999999\"><< anterior</font> |";
  }else{
    print "<< <a href=\"index.php?pagina=".($pagina - 1)."\" targe=\"_self\">anterior</a> |";
  }

  $x = 1;
  while ($x <= $total_paginas) {
  if ($x == $pagina){
      print " <b><u>$x</u></b> ";
    }else{
      print " <a href=\"index.php?pagina=".$x."\" target=\"_self\">".$x."</a> ";
    }
    $x = $x + 1;
  }

  if ($pagina == $total_paginas) {
    print "| <font color=\"#999999\">próxima >></font>\n";
  }else{
    print "| <a href=\"index.php?pagina=".($pagina + 1)."\" targe=\"_self\">próxima</a> >>\n";
  }

  mysql_free_result($resultado);
  mysql_close($conexao);
?>
```

imprimir os dados. Veja:

```
print "  <td bgcolor=$cor>" .
  $row['ID'] . "</td>";
print "  <td bgcolor=$cor>" .
  $row['TITULO'] . "</td>";
print "  <td bgcolor=$cor>" .
  $row['ISBN'] . "</td>";
print "  <td bgcolor=$cor>" .
  $row['AUTOR'] . "</td>";
print "  <td bgcolor=$cor>" .
  $row['GENERO'] . "</td>";
```

Observe especificamente o item *$row*. Ele é um *array* que está sendo incrementado no laço *while* e por esse motivo contém os registros da base de dados:

```
while ($row =
  mysql_fetch_array($resultado)) {
```

Agora a parte final. Aqui na **Listagem 4** apenas estamos imprimindo a numeração e os links para as páginas de livros. Também é código *HTML* puro. Nosso propósito final é termos algo parecido com o que vemos a seguir:

```
<<anterior 1 2 3 próxima >>
```

Ou seja, um verdadeiro navegador. Implemente o código e finalize com o fechamento da conexão e limpeza da referência à variável:

```
mysql_free_result($resultado);
mysql_close($conexao);
```

A **Listagem 5** contempla todo o código-fonte da página que montamos. Salve esse arquivo como *index.php* e execute em seu *browser* de preferência. Veja o resultado na **Figura 13**.

Conclusão

Neste artigo vimos como criar facilmente um sistema de paginação para nosso *Web Site*. Em paralelo acabamos vendo como fazer a instalação do *MySQL* e de uma ferramenta de administração para esse banco. Certamente ainda há muito que se falar sobre PHP e seus macetes, mas com todo conteúdo visto aqui, já é possível desenvolver grandes sites. Mão na massa e até mais. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

Faça já sua assinatura digital! | *www.devmedia.com.br/es*

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia GROUP

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet